Final Report

SSME LOX POST FLOW ANALYSIS / FLUID STRUCTURE INTERACTION

Volume II: Fluid Structure Interaction

June 1989

Contract NAS8-37361

Prepared for

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
GEORGE C. MARSHALL SPACE FLIGHT CENTER
MARSHALL SPACE FLIGHT CENTER, ALABAMA 35812

by

The Computational Mechanics Co., Inc.
Austin, TX 78705

Lockheed
Missiles & Space Company, Inc.
Huntsville Engineering Center
4800 Bradford Blvd., Huntsville, AL 35807

Final Report

SSME LOX POST FLOW ANALYSIS / FLUID STRUCTURE INTERACTION

Volume II: Fluid Structure Interaction

June 1989

Contract NAS8-37361

Prepared for

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
GEORGE C. MARSHALL SPACE FLIGHT CENTER
MARSHALL SPACE FLIGHT CENTER, ALABAMA 35812

by

The Computational Mechanics Co., Inc.
Austin, TX 78705

Lockheed Missiles & Space Company, Inc.
Huntsville Engineering Center
Huntsville, AL 35807

Contents

1 INTRODUCTION

A growing number of problems in the field of engineering require the study of fluid–structural systems exposed to complex dynamic loadings. In particular, the deformation and yielding of lox posts in the Space Shuttle Main Engine and the eroding of solid propellant boundaries in the Solid Rocket Boosters are known to have significant influence on the flow field structure and on performance of the systems. The modeling of such phenomena couples a dynamic structural analysis with a transient nonlinear fluid analysis for which the computational domain is continuously changing due to structural motions. New modeling and computational techniques are needed to simulate these types of fluid–structure interactions (FSI) in order to factor such effects into design considerations.

This report documents the progress made in resolving the computational issues associated with modeling high temperature and pressure viscous flows in advanced propulsion systems. The first phase of this effort has been devoted to the development and testing of various adaptive techniques for dynamically updating a computational domain to reflect the deformations of various structural components. Results from applying these adaptive procedures to two example problems, the lox-post problem and the flexible cylinder problem, clearly demonstrate the potential of the methods under development.

The following sections provide specific details on the work completed during the first phase of this project. The first section presents an overview of adaptive computational methods for various classes of fluid–structure interaction problems. Here the key features associated with several approaches are outlined and the advantages/disadvantages of each are summarized. Section 3 presents a mathematical formulation for general fluid–structure interaction problems with a moving domain. The next section discusses the two adaptive approaches used in solving the benchmark problems presented in Section 5. The first scheme is a user interactive scheme which is quite versatile and easy to implement but requires an excessive amount of monitoring by the operator and is computationally inefficient. The second method is a new, local remeshing method for handling a general class of fluid–structure interaction problems. This method couples many of the attractive features of other approaches into a computationally efficient and versatile method. The following section, Section 5, presents the results obtained for two test cases. Results from both the user interactive and local remeshing procedure are described. The final section summarizes the current status of the project and future goals to be completed.

2 LITERATURE SURVEY ON MOVING MESH ALGORITHMS

2.1 Automatic Mesh Displacement Method

The automatic mesh displacement method is based on the paper presented by Donea, Giuliani, and Halleux [1]. Because of the shortcomings of purely Lagrangian and purely Eulerian descriptions, efforts have recently been expanded in the finite element area to develop integrated procedures with generalized kinematical descriptions of the fluid domain that possess both Eulerian and Lagrangian features. These generalized descriptions are generally referred to as arbitrary Lagrangian-Eulerian (ALE) algorithms. The methodology employed with this approach is to describe an ALE finite element formulation with automatic and continuous rezoning of the fluid mesh. The key steps involved here are as follows:

1. An arbitrary Lagrangian-Eulerian (ALE) kinematical description of the fluid domain is adopted in which the grid points can be displaced independently of the fluid motion.

2. In the automatic mesh displacement prescription algorithm, grid velocity is computed at each step of the time integration procedure.

$$w_{I,i}^{t+\Delta t} = \frac{1}{N}\sum_J w_{J,i}^t + \frac{0.1}{\Delta t}\sum_J L_{IJ}^t \sum_J \frac{\delta_{J,i}^t - \delta_{I,i}^t}{L_{IJ}^t} \cdot \frac{1}{N^2} \tag{2.1}$$

   where

   i $\sim$ components of a vector
   I, J $\sim$ node numbers
   N $\sim$ the number of nodes connected to node I via element sides and diagonals
   L_{IJ} $\sim$ the current distance between node I and connected node J
   δ $\sim$ total nodal displacements
   Δt $\sim$ time step

3. The second term of the right hand side of Eq. (2.1) is a corrective term which enhances the grid velocity when the distance between adjacent nodes tends to become too short. This will keep the fluid mesh regular.

A plot depicting this approach is shown in Fig. 2.1. Advantages and disadvantages are listed below:

1. Advantages

   (a) Internal fluid nodes are moved automatically by the program

   (b) ALE description allows the fluid mesh to remain regular

(c) Method consists of a continuous rezoning process, thus transient and steady-state modeling is feasible.

2. Disadvantages

(a) Calculates the grid velocity for all nodes (globally), i.e., grid motion is not restricted to a selected region and thus additional CPU time is required for regions of the mesh where grid motion is negligible.

(b) Limited flexibility for large structural motions

2.2 Overlapping Grid Method

The Chimera grid scheme is one of the few methods found in the literature search that does not require regridding of the mesh. This method, an overlapping grid method, consists of generating a main grid over the entire flow domain with smaller grids generated over areas of interest in the flow field. Each grid is solved separately using information from the other grids. Minor grids use interpolated data from the major grid for specifying far-field boundary conditions while the major grid has the points that overlap nearest to the center of the minor grid "blanked out." The values of these blanked points are interpolated from the solution of the minor grid after the major grid iteration is complete.

Four major changes are necessary to adapt a single grid scheme to a multi-grid scheme. First, the database structure must be altered to manage multiple grids of different dimensions. Second, the "blanked" points of the major grid must be recognized, flagged, and properly handled. Third, the boundary conditions need to be adjusted so that each grid can have its boundary conditions separately specified. Finally, interpolation routines are required. Steger, Dougherty, and Benek [2, 3] have implemented these changes and have shown good results for solutions to the potential flow and Euler equations.

The advantages of using a multigrid routine include the following:

1. Complex geometry problems can be solved without difficulties with grid distortion.

2. The grids are easy to contruct and are well-ordered.

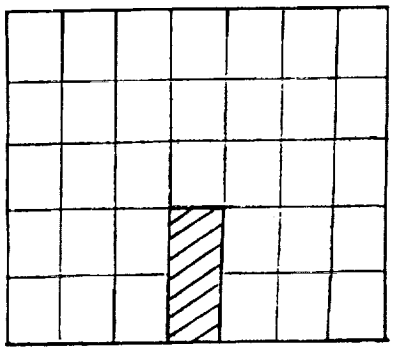
3. The overlapping of the grids provides a "nice" net for interpolation.

4. Separate equation systems can be solved on each grid. For example, the Euler equations can be solved on a minor grid around an airfoil with a potential flow solution in the major grid.

Some of the disadvantages of such a scheme are:

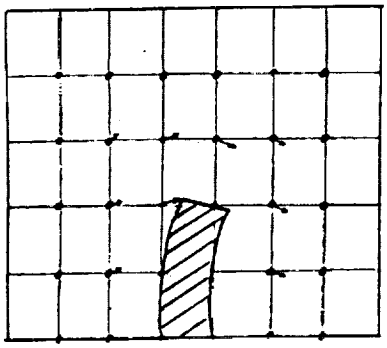
1. Labeling of blanked out points must be handled separately.

(1)



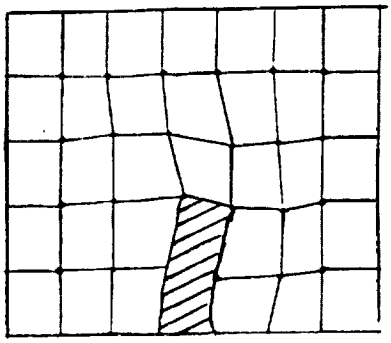
Initial Grid

(2)



Grid velocity is computed
at each time step.

(3)



Grid will be updated automatically
at each time step.

Figure 2.1: Schematic of the automatic mesh displacement or ALE (arbitrary Lagrangian-Eulerian) method.

2. An interpolation procedure between overlapping grids must be implemented.

3. Multiple interpolated boundary conditions must be handled.

4. Additional vectors and storage are required to handle multiple grids and boundary conditions.

Some sample domains from Steger, Dougherty, and Benek [2] are shown in Figs. 2.2 and 2.3.

2.3 Background Grid/Moving Mesh Method

The methodology for this algorithm, based on investigations by Atluri and Nishioka [4], consists of dividing the domain into three types of elements: moving elements, distorting regular elements, and non–distorting regular elements. The bulk of the domain is discretized by the non–distorting elements. The areas of the mesh where motion is occurring are locally discretized by moving elements with distorting regular elements adjacent to the moving ones. These distorting regular elements, therefore, serve as as a "buffer" between the moving elements and the non–distorting elements. Shown in Fig. 2.4 is a typical deformation of a simple structural component. (For simplicity's sake, the moving elements will be designated "A", the distorting regular elements "B", and the non–distorting regular elements "C".) When an A element moves, the B elements become distorted as the nodes common with C elements remain fixed while the nodes common with A elements move. The mesh is readjusted before the B elements become too distorted, i.e. after the moving element has traversed half a side length. The adjustment involves a shift in the connectivity between the B and C elements. The result of this procedure is the creation of C elements and smaller B elements behind the motion and the change from small B and C elements to large B elements ahead of the motion.

This method appears attractive as the grid motion is localized and much of the domain remains stationary and regular. Problems arise, though, when this model is extended beyond "two-element-deep" domains. For example, consider the initial truncated domain in Fig. 2.5a. The A elements form an elastic–type beam that is subjected to forces causing it to bend. Hypothetical beam displacements are shown in Figs. 2.5b–d with their respective meshes. The meshes of Figs. 2.5b and 2.5c show the distortion of the B elements as the A elements move. In Fig. 2.5d, one sees the generation and destruction of C elements due to the changing connectivity of the B elements. In addition, there is a problem arising that does not occur with two–deep element problems: the generation of five–node elements.

The five–node element problem can be eliminated by altering the Atluri-Nishioka method. Instead of using distorting four–node elements as "buffer" elements, a mesh of triangular elements around the moving elements can be generated. Preliminary versions of this triangular remeshing are shown in Figs. 2.6a and 2.6b.

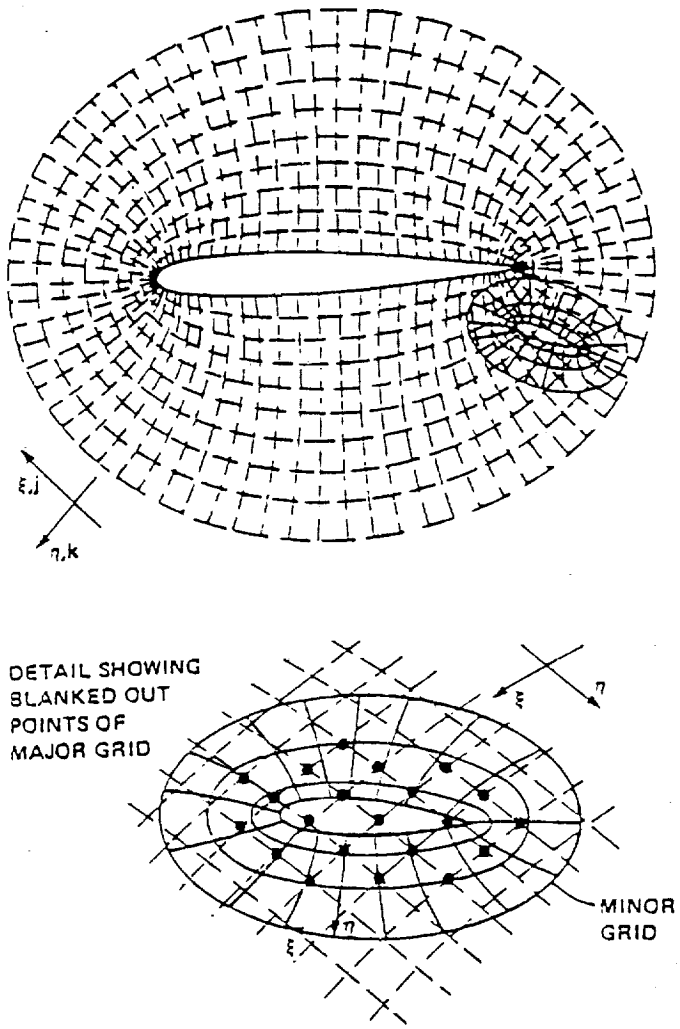


Figure 2.2: Some sample grids for the Chimera scheme suggested by Steger, Dougherty, and Benek [2].

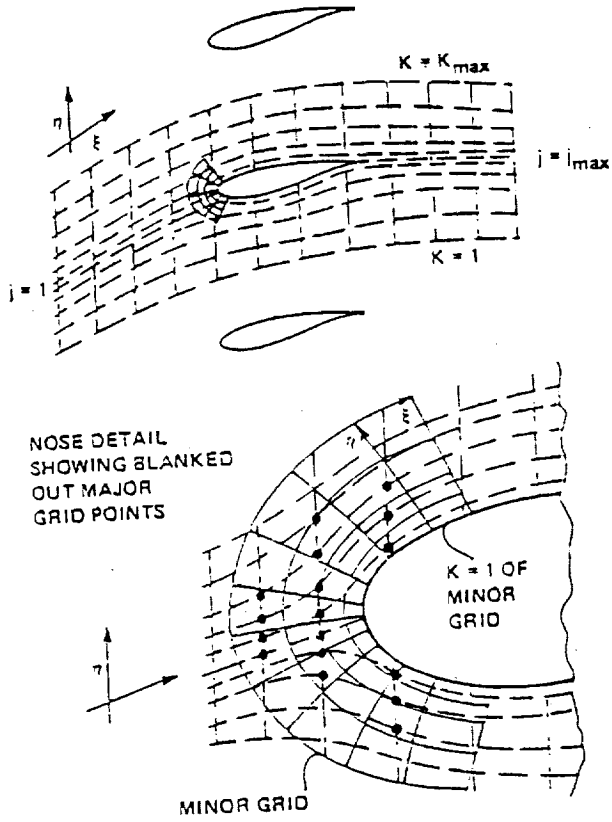


Figure 2.3: Some sample grids for the Chimera scheme suggested by Steger, Dougherty, and Benek [2].

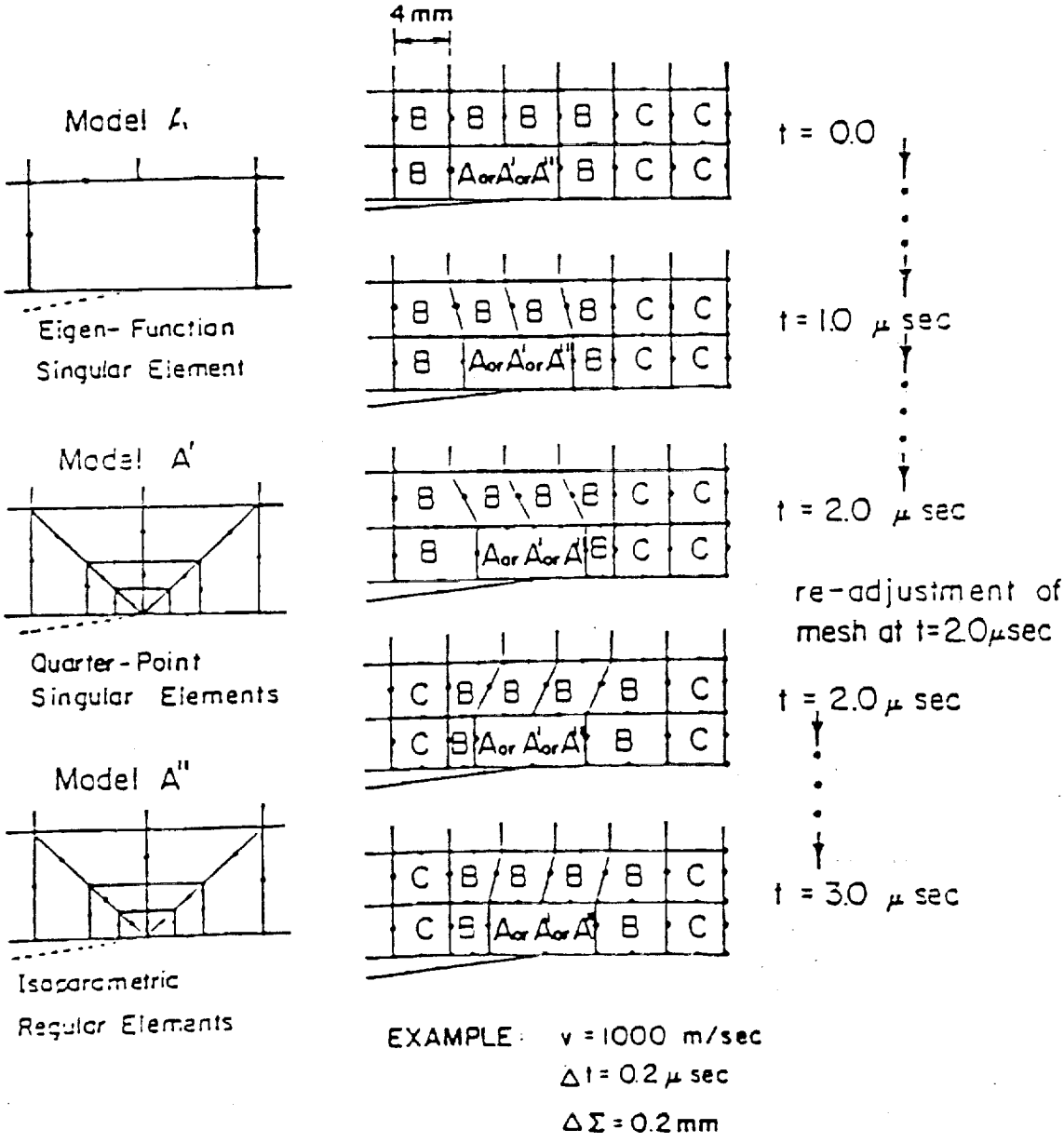


Figure 2.4: Grid motion suggested by Atluri and Nishioka [4]. This example is for the fracture mechanics problem of a propagating crack.

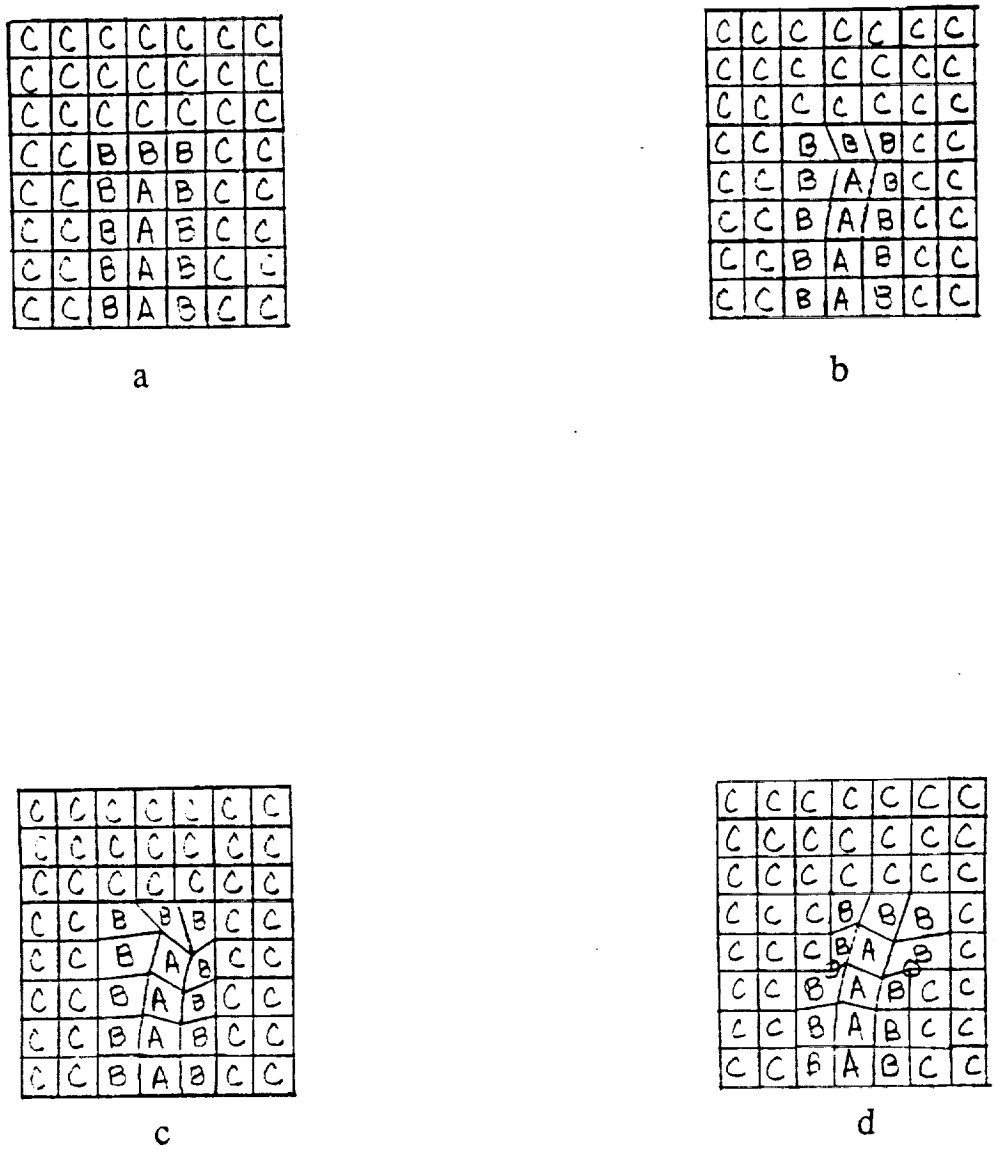
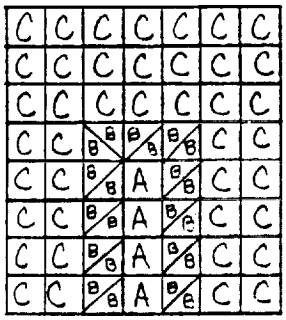
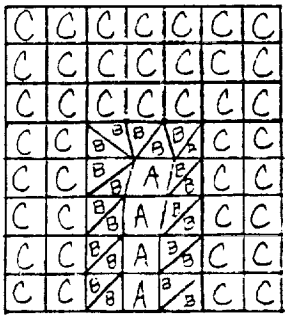


Figure 2.5: Atluri and Nishioka method

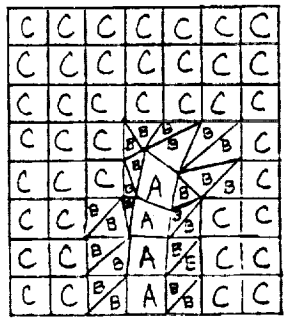
The method of Atluri and Nishioka has been extended beyond two elements here. The A elements represent an elastic beam. The deformation of the grid is acceptable in Figs. 2.5a–c, but five–node elements are generated by the deformation shown in Fig. 2.5d.



Initial Grid



a



b

Figure 2.6: Atluri and Nishioka method, modified.

The method of Atluri and Nishioka is modified to include triangular elements as buffer elements. The problem of the five–node elements has been eliminated.

2.4 Other Methods

Lynch [5] suggests two remeshing equations. The first is a simple proportional stretching:

$$\boldsymbol{x}_i(t) = \boldsymbol{s}(t)[\boldsymbol{x}_i(0)/\boldsymbol{s}(0)] \tag{2.2}$$

where $\boldsymbol{x}_i$ is the location of node i and $\boldsymbol{s}$ is the location of the moving boundary node "above" it. This method assumes that the boundary motion is in one direction only.

A second method assumes that the grid is an elastic material that satisfies the following equations:

$$\begin{aligned} \frac{\partial^2 \dot{U}}{\partial x^2} + \frac{\nu \partial^2 \dot{V}}{\partial x \partial y} + \left(\frac{1-\nu}{2}\right)\left[\frac{\partial^2 \dot{U}}{\partial y^2} + \frac{\partial^2 \dot{V}}{\partial x \partial y}\right] &= 0 \\ \frac{\partial^2 \dot{V}}{\partial y^2} + \frac{\nu \partial^2 \dot{U}}{\partial x d \partial y} + \left(\frac{1-\nu}{2}\right)\left[\frac{\partial^2 \dot{V}}{\partial x^2} + \frac{\partial^2 \dot{U}}{\partial x \partial y}\right] &= 0 \end{aligned} \tag{2.3}$$

where $(\dot{U}, \dot{V})$ is the rate of displacement of a point in the mesh and ν is Poisson's ratio. If the movement of the boundary is prescribed, the velocities at the interior nodes can be found by solving the elastic equations. The displacement of the grid is then calculated as:

$$\boldsymbol{x}_i^{t+\Delta t} - \boldsymbol{x}_i^t = \Delta t[\theta \boldsymbol{v}_i^{t+\Delta t} + (1-\theta)\boldsymbol{v}_i^t] \tag{2.4}$$

where θ is a time weighting parameter.

For regular mesh generation, Jacquotte [6] minimizes a functional derived from element characteristics. For two dimensions, the following functional is suggested:

$$\begin{aligned} F &: \alpha \sum_e SM^e + (1-\alpha) \sum_e ORT^e \\ SM^e &= (\boldsymbol{r}_1^2 + \boldsymbol{r}_2^2 + \boldsymbol{r}_3^2 + \boldsymbol{r}_4^2)^e \\ ORT^e &= [(\boldsymbol{r}_1 \cdot \boldsymbol{r}_2)^2 + (\boldsymbol{r}_2 \cdot \boldsymbol{r}_3)^2 + (\boldsymbol{r}_3 \cdot \boldsymbol{r}_4)^2 + (\boldsymbol{r}_4 \cdot \boldsymbol{r}_1)^2]^e \end{aligned} \tag{2.5}$$

with $\boldsymbol{r}_1$ the directed vector from local node 1 to node 2 of an element e and α a parameter that emphasizes the smoothness (SM^e) or orthogonality (ORT^e) function.

In three dimensions, the functional σ is introduced. σ is derived from invariants of the Cauchy-Green tensor of the deformation. Specifically, if $\boldsymbol{x}$ are the coordinates of the grid, the Cauchy-Green tensor is defined as:

$$C = \nabla x^T \cdot \nabla x \tag{2.6}$$

The functional is then given by $\sigma = \sigma(I_1, I_2, I_3)$ where I_1 is the trace of C, I_2 is the trace of the cofactor matrix of C, and I_3 is the determinant of C.

For the two–dimensional scheme, the nodes of two adjacent sides of an element are fixed while the nodes on the other sides are allowed to move in order to minimize the functional. In three dimensions, three adjacent sides of the element are fixed.

An adaptive algorithm that combines both remeshing and refining methods of grid adaptation is based on studies by Kikuchi and Cheng [7]. Much of this method depends on the selection and definition of an error estimator. This error estimator is based on the maximum value over the entire grid of the error associated with each element. (Often error estimates are based on cumulative error.) The defining function of the error can be chosen for a particular problem (i.e., energy, strain). Once the error estimate has been calculated, it is then used as a weighting function for the remeshing algorithm as follows:

$$\boldsymbol{X}_n = (\sum_e \boldsymbol{X}_e(E_e/A_e))/\sum_e(E_e/A_e)) \tag{2.7}$$

where $\boldsymbol{X}_n$ is the new location of node n, $\boldsymbol{X}_e$ is the geometric centroid of element e, A_e is the area of the element, and E_e is the error indicator of the element. The summation is taken over the elements attached to node n. The result of the remeshing process is a more even distribution of the error over the new elements. The error estimate may also be used as a refining criterion in order to determine if new elements are to be generated.

3 FORMULATIONS OF FLUID–STRUCTURE INTERACTION PROBLEMS

3.1 Weak Formulation of Fluid–Structure Interaction Problems With Moving Domains

Consider a region Ω in $\mathbb{R}^3$ through which a material body, fluid or solid, moves. A fixed spatial frame of reference can be established with origin O and position vectors emanating from O denoted by $\boldsymbol{x}$. Cartesian components of $\boldsymbol{x}$ are denoted x_i, $i = 1, 2, 3$. A material particle moving through Ω is labeled X; it is always possible to associate with X an ordered triple $\boldsymbol{X} = (X^1, X^2, X^3)$ of material coordinates which, for example, coincide with the coordinates x_i of the particle in some reference configuration of the medium. The motion of the material body is then defined by a one-parameter family of mappings given the spatial positions of particles as functions of time:

$$\boldsymbol{x} = \boldsymbol{\chi}(\boldsymbol{X}, t)$$

The particle velocity $\boldsymbol{u}$, relative to the spatial frame, is then given by the material time derivative of the motion (keeping X fixed):

$$\boldsymbol{u} = \frac{D\boldsymbol{x}}{DT} = \frac{\partial \boldsymbol{\chi}}{\partial t}$$

The acceleration is then

$$\boldsymbol{a} = \frac{D\boldsymbol{u}}{DT} = \frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \boldsymbol{\nabla} \boldsymbol{u}$$

where $\boldsymbol{\nabla}$ is the spatial gradient operator ($\boldsymbol{\nabla} \boldsymbol{u} = \partial u_i / \partial x \cdot \boldsymbol{i}_i \otimes \boldsymbol{i}_i$) with the summation convention in force).

The mechanical behavior of the medium is governed by the conservation/balance laws of physics and by the second law of thermodynamics. The balance laws can be written in terms of primitive or conservation variables which can be arranged in a vector $\boldsymbol{U} = \boldsymbol{U}(\boldsymbol{x}, t)$; for example, in terms of conservation variables, for a compressible fluid, we have

$$\boldsymbol{U} = \{\rho, \boldsymbol{m}, e\}^T$$

where ρ is the mass density, $\boldsymbol{m}$ the momentum vector, and e the energy. The conservation/balance laws can thus be written

$$\frac{d}{dt} \int_\Omega \boldsymbol{U} dx = - \int_{\partial\Omega} \boldsymbol{Q} \cdot \boldsymbol{n} ds + \int_\Omega \boldsymbol{B} dx \tag{3.1}$$

where dx and ds are volume and surface area measures of Ω and its boundary $\partial\Omega$, respectively, $\boldsymbol{Q}$ is the flux matrix, and $\boldsymbol{B}$ is a vector of source terms. In three–dimensional models, (3.1) represents five equations: the continuity equation representing the conservation of mass, three equations expressing the balance of linear momentum, and the final equation describing the conservation of energy.

The flux can generally be written as the sum of two parts. For three–dimensional problems,

$$Q_{\alpha i} = u_i U_\alpha + \Sigma_{\alpha i} \ , \ i = 1,2,3 \ \ \alpha = 1,2,3,4,5$$

where

$$\begin{aligned} U_1 &= \rho, \ U_\alpha = m_{\alpha-1} (\alpha = 2,3,4), \ U_5 = e \\ \Sigma_{1i} &= 0; \ \Sigma_{\alpha i} = \sigma_{\alpha-1,i} \ , \ \alpha = 2,3,4, \\ \Sigma_{5i} &= u_j \sigma_{ij}, \ i,j = 1,2,3 \end{aligned}$$

where σ_{ij} are Cartesian components of the Cauchy stress tensor, m_i are the components of the momentum vector, and σ_{ij} is symmetric, by virtue of the principle of balance of angular momentum. Expressing the equations in component form results in the following:

$$\begin{aligned} \frac{d}{dt} \int_\Omega U_\alpha dx &= -\int_{\partial\Omega} (u_i U_\alpha n_i + \Sigma_{\alpha i} n_i) ds \\ &+ \int_\Omega B_\alpha dx \end{aligned} \tag{3.2}$$

with repeated indices summed and $1 \leq \alpha \leq 5$, $1 \leq i \leq 3$.

A weak formulation of the problem is easily obtained in the usual way from (3.1):
Find U_α in a class of trial functions V such that

$$\int_0^T \int_\Omega \frac{\partial U_\alpha}{\partial t} \phi_\alpha dx dt = -\int_0^T \int_{\partial\Omega} (u_i \phi_\alpha U_\alpha n_i + \Sigma_{\alpha i} \phi_\alpha n_i) dx dt + \int_0^T \int_\Omega B_\alpha \phi_\alpha dx dt \tag{3.3}$$

$$\forall \phi_\alpha \text{ in a class } W \text{ of test functions}$$

where the repeated index α is summed from 1 to 5.

Now consider a moving system of coordinates y attached to a volume G which moves through Ω. The coordinates X_i, x_i, and y_i are related through a transformation

$$\boldsymbol{y} = \overline{\Psi}(\boldsymbol{X}, t) = \overline{\Psi}(\chi^{-1}(\boldsymbol{x}, t), t) = \boldsymbol{\Psi}(\boldsymbol{x}, t)$$

The velocity of this frame relative to the fixed spatial frame of reference is

$$\boldsymbol{u}^G = \frac{\partial \overline{\Psi}}{\partial t}$$

Since the only way that matter can be carried through the boundary ∂G of G is to attain a velocity which exceeds the local particle velocity, instead of (3.3), the general formulation should read

$$\begin{aligned} \int_0^T \int_G \frac{\partial U_\alpha}{\partial t} \phi_\alpha dy dt &= -\int_0^T \int_{\partial G} (u_i - u_i^G) \phi_\alpha U_\alpha n_i ds dt - \int_0^T \int_{\partial G} \Sigma_{\alpha i} \phi_\alpha n_i ds dt \\ &+ \int_0^T \int_G B_\alpha \phi_\alpha dy dt \qquad \forall \phi_\alpha \text{ in } W \end{aligned} \tag{3.4}$$

where it is understood that the integrands are functions of the moving grid coordinates y_i. In fluid–structure interaction problems, the key issue is determining an appropriate grid velocity $\boldsymbol{u}^G$ for the problem at hand. It is clear, however, that formulation (3.4) can apply to either solid bodies, fluids, or to rigid bodies moving through fluids, with an appropriate definition of the constitution of the material. For example, for small displacement gradients $w_{ij} = \partial w_i / \partial X^j$ of a Hookean solid, the Cauchy stress is expressed as

$$\sigma_{ij} = E_{ijk\ell} w_{k,\ell} \tag{3.5}$$

where $E_{ijk\ell}$ is the array of material elastic constants. For an ideal inviscid compressible gas,

$$\sigma_{ij} = -p(\boldsymbol{U})\delta_{ij} \tag{3.6}$$

where $p(\boldsymbol{U})$ is the thermodynamic pressure,

$$p(\boldsymbol{U}) = (\gamma - 1)(e - \rho^{-1}\boldsymbol{m} \cdot \boldsymbol{m}/2) \tag{3.7}$$

γ being the ratio of specific heats, and for an incompressible viscous fluid,

$$\sigma_{ij} = -p\delta_{ij} + \frac{\mu}{2}\left[\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right] \tag{3.8}$$

where p is the hydrostatic pressure and μ is the fluid viscosity.

3.2 Discrete Formulations

Discrete models of weak formulations are constructed on finite–dimensional subspaces V^h of V and W^h of W, which here are constructed using piecewise polynomial approximations over a mesh Ω_h (or G_h) of elements Ω_e, $1 \le e \le E$. In general,

$$U_\alpha \approx U_\alpha^h,\ U_\alpha^h|_{\Omega_e} \in P_k(\Omega_e)$$

where $P_k(\Omega_e)$ is the space of polynomials of degree $\le k$ defined on Ω_e. If a strip $D_n = \Omega_e \times [t_1, t_2] \subset \Omega \times [0, T]$ is also considered, then instead of (3.4), the problem now consists of finding $\boldsymbol{U}^h$ such that

$$\begin{aligned}\int_{t_1}^{t_2}\int_{\Omega_e} U_\alpha^h \frac{\partial \phi_\alpha^h}{\partial t} dy dt = &- \int_{t_1}^{t_2}\int_{\partial\Omega_e} [(u_i^h - u_i^G)\phi_\alpha^h U_\alpha^h n_i \\ &- \Sigma_{\alpha i}(\boldsymbol{U}^h)\phi_\alpha^h n_i] ds dt + \int_{t_1}^{t_2}\int_{\Omega_e} b_\alpha \phi_\alpha^h dy dt \end{aligned} \tag{3.9}$$

$$\forall \phi_\alpha^h \in W^h$$

The semi–discrete problem (3.9) is, of course, only a formal statement of a broad class of finite element formulations. Specific algorithms are obtained from (3.9) by specifying quadrature rules over $[t_1, t_2]$, choosing local shape functions, specifying $\boldsymbol{u}^G$ or an algorithm for determining it, and incorporating the solution strategy into an appropriate adaptive scheme.

3.3 Boundary Conditions

The integration by parts formulas used to obtain the weak formulation lead to two boundary integral contributions as shown on the right–hand side of (3.4). For the most part these are typical boundary integrals found in the Navier-Stokes equation with the Euler fluxes replaced by $\boldsymbol{E} - u_G \boldsymbol{U}$ and $\boldsymbol{F} - v_G \boldsymbol{U}$. The essential difference is embodied in the additional work term done by the pressure on the moving boundary. These are given by

$$\begin{aligned} &-\int_0^t \int_{\partial\Omega} p \boldsymbol{n}_{xi} \phi_\alpha ds dt \\ &-\int_0^t \int_{\partial\Omega} p \boldsymbol{u}_G \cdot \boldsymbol{n} \phi_\alpha ds dt \end{aligned} \tag{3.10}$$

for the momentum and energy equations, respectively. Here $\boldsymbol{n}_{xi}$ is the outward unit normal vector.

4 ADAPTIVE METHODS FOR FLUID-STRUCTURE INTERACTION APPLICATIONS

One of the primary goals of the project is to develop a computational methodology which is capable of dynamically handling arbitrarily complex motions of bodies or boundaries in a fluid domain. The literature survey presented in Section 2 was the first step toward the development of such a methodology. After careful analysis of the published research, two approaches have been selected for further study. The first, a quasi-steady-state method, was chosen based on the ease of implementation and compatability with the class of problems being benchmarked. The second, a local remeshing method, was selected for its flexibility in modeling a large class of fluid–structure interaction problems and for its computational efficiency. Details for each of these algorithms are provided in this section.

4.1 Quasi-Steady-State Method

The quasi-steady-state method (QSSM) combines a grid generation scheme with a flow solver for the Navier-Stokes equations and a solid mechanics equation solver in a user–interactive iterative sequence to obtain steady–state solutions for fluid–structure interaction problems. The specific steps involved in the method are as follows:

1. Generate an initial computational grid for the fluid domain.
2. Solve the viscous flow problem using the grid generated in step 1 for steady–state conditions.
3. Extract the nodal values of the pressure and viscous stress from the fluid solution on the boundary of the deformable structure.
4. Solve the structural problem subjected to specified surface tractions to determine the deformed configuration.
5. If a convergence tolerance of the structural motion is satisfied, STOP.
6. Generate a new fluid grid using the results from step 4.
7. Go to step 2.

At the present time, this sequence of steps requires a great deal of user–interactive input as the modules for the grid generation, flow solver, and structural modeling are all completely separate. However, such an approach does possess a great deal of flexibility in that any component of the solution process can be altered without significantly affecting the results in the other areas.

For the first step in this sequence, a new grid generation scheme was devised. This scheme is based on the parabolic partial differential equation grid generation method suggested by Nakamura in [10]. This approach is constructed by modifying the elliptic generation system

$$\begin{aligned} Ax_{\xi\xi} + Bx_{\xi\eta} + Cx_{\eta\eta} &= 0 \\ Ay_{\xi\xi} + By_{\xi\eta} + Cy_{\eta\eta} &= 0 \end{aligned} \tag{4.1}$$

$$A = x_\eta^2 + y_\eta^2, B = -2(x_\xi x_\eta + y_\xi y_\eta), C = x_\xi^2 + y_\xi^2$$

such that the second derivatives in one coordinate direction do not appear. The solution can then be marched away from the boundary in much the same manner as for hyperbolic systems.

This parabolic method of grid generation incorporates many of the advantages of both elliptic [8] and hyperbolic [9] methods including:

1. The grids are generated by a marching algorithm similar to the hyperbolic grid generation methods.
2. The parabolic differential equations exhibit a diffusion effect which smooths out singularities in the inner boundary conditions as exhibited by the elliptic generations methods.
3. Prescribed outer boundary conditions may be satisfied.

In addition to these advantages, this approach requires significantly less storage and computational time than elliptic methods and avoids the boundary condition and stability problems associated with hyperbolic methods.

The second step in the quasi-steady-state method involves obtaining the viscous solution for the fluid grid generated in step 1. For this step the Taylor-Galerkin two–step algorithm operational in an existing in–house finite element code was used. The third step was simply a post–processing step which extracted the nodal values of the pressure and viscous stresses. Step four was accomplished using a simple linear elasticity code for solving planar problems. The subsequent steps are iterations of the preceding steps with different specified initial and boundary conditions.

To demonstrate the capabilities of the parabolic grid generator, initial grids for the two benchmark problems are presented here. In the first example, the flexible wall problem, an H-grid consisting of 2400 elements was constructed as shown in Fig. 4.1. In the generation of this mesh, a cluster function

$$y = \overline{A}[x + c\sin(w\pi x/W)]$$

was used to stretch the grid in the region of the deformable wall. Here, $\overline{A}$ is an amplitude parameter, x is the original position, y is the computational position, c is the cluster parameter, and w is the wave number. The scale in the figure has been nondimensionalized by the length of the wall using the following dimensions:

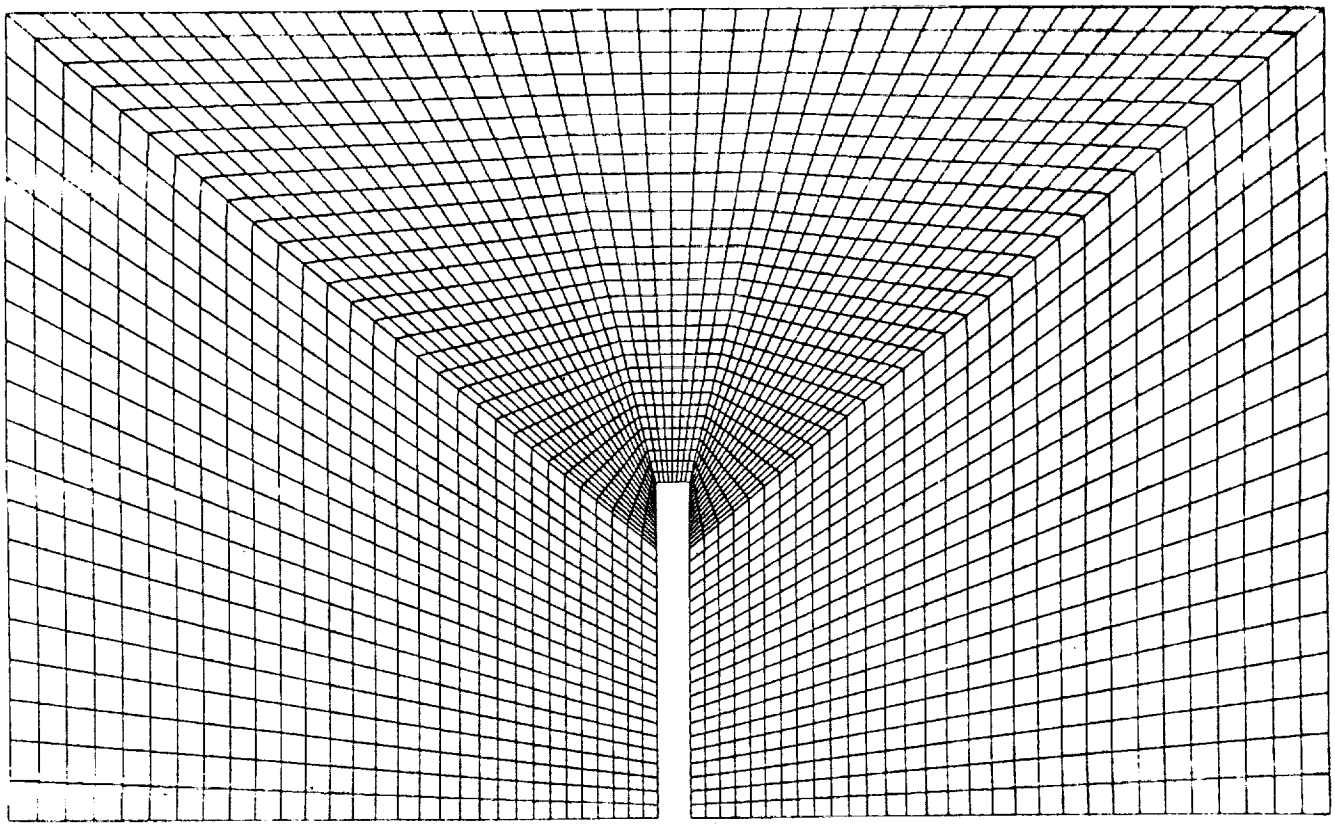


Figure 4.1: Initial grid generated for the flexible wall problem using the quasi-steady state method.

Wall height	=	1.0
Wall thickness	=	0.1
Length of computational domain	=	4.0
Height of computational domain	=	2.4

A similar procedure was used to generate an initial grid for the rigid cylinder problem, shown in Fig. 4.2. In this figure, the structure has been decomposed into three sections: section A is the rigid section of the shaft with length R, section B is the flexible section of the shaft with length 5R, and section C is the rigid cylinder. The thickness of the shaft is constant at 0.25R and the cylinder radius is specified as R = 1.0. The computational fluid domain consisting of 2520 elements was obtained using a cluster parameter of $c = 0.0$.

4.2 Local Remeshing Method

Although there is a wide range of motion–related fluid–structure interaction problems, it is believed that the majority fall into one of three general problem classes. The first class consists of fixed yet flexible body motion such as the motion of a cantilevered beam. The second class describes a free boundary variable domain problem. The third class of problems are concerned with free body motion relative to a fixed point. The motion of a projectile relative to the point of discharge is an example in this final set.

For most problems, any motion will occur in a restricted region of the domain, a so-called "region of influence." By defining this region of influence, problem formulations and algorithms can be simplified by restricting motion formulations to a subdomain and thereby allowing simpler static formulations to be implemented outside of this region. Suitable regions of influence for the three problem classes cited above include the following:

Class I	Area encompassing the expected deflection of the cantilevered beam
Class II	Banded region about a moving boundary
Class III	Domain fixed to the frame of reference of the projectile

Samples of the three problem classes and regions of influence are presented in Fig. 4.3. The recognition of these regions of influence are important for it will be in these areas where local remeshing will occur.

The local remeshing method is a combination of a number of the algorithms presented in the literature. It combines the ideas of node relocation, remeshing and overlapping grids into a single methodology for handling FSI problems. The steps involved in this method are as follows:

1. Generate initial background grids for the fluid and the structure. (Presently these are rectangular grids.)

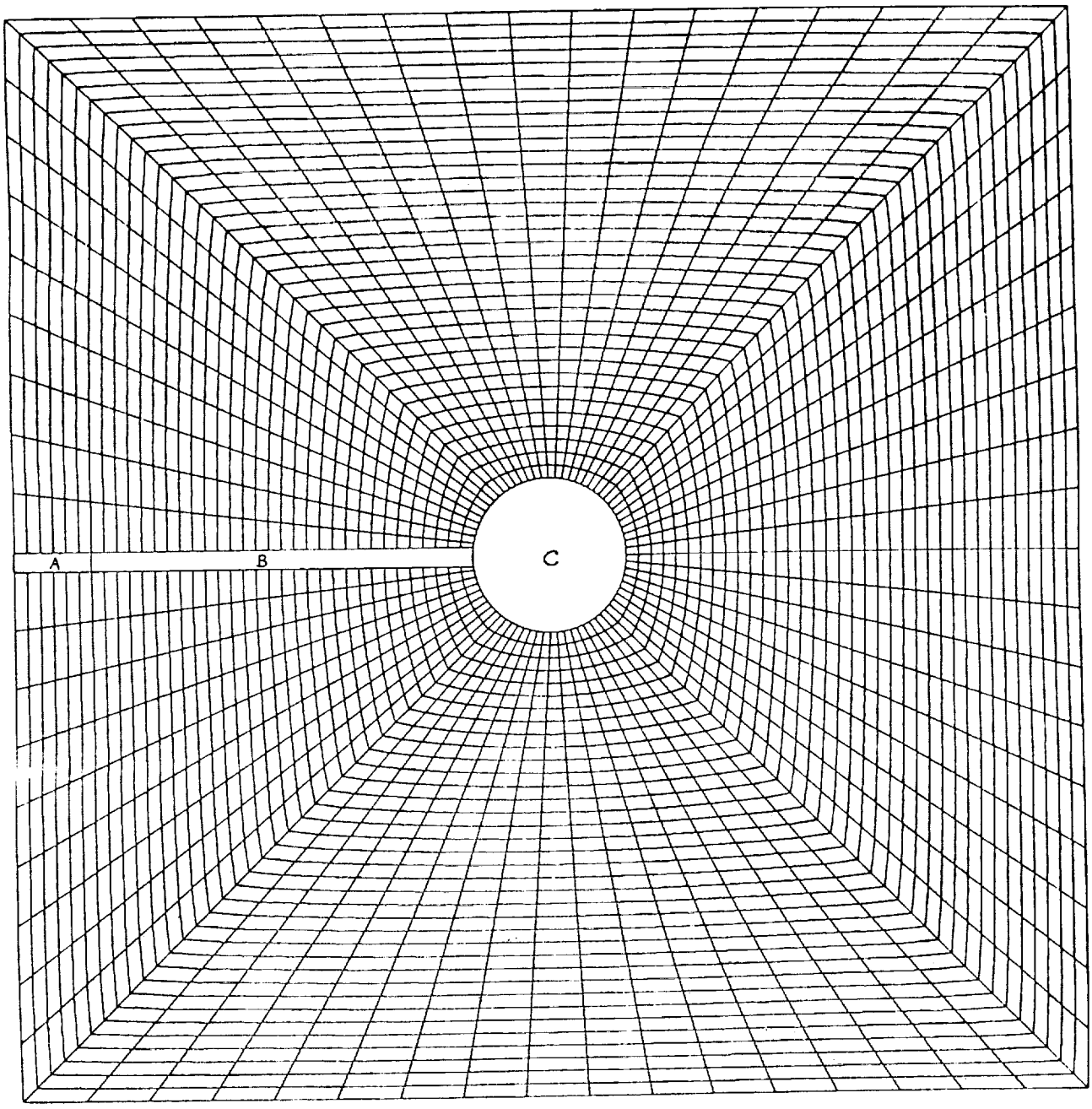
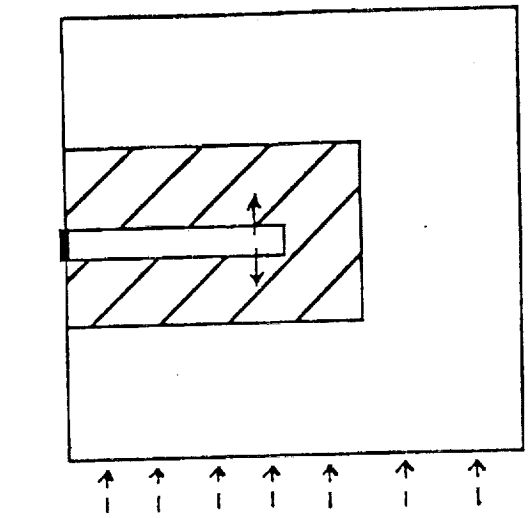


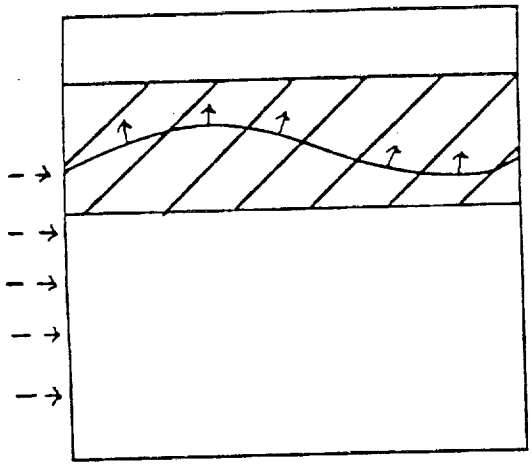
Figure 4.2: Initial grid generated for the problem of a rigid cylinder on a flexible shaft for use with the quasi–steady state solution method.



CLASS I

Fixed Object Motion

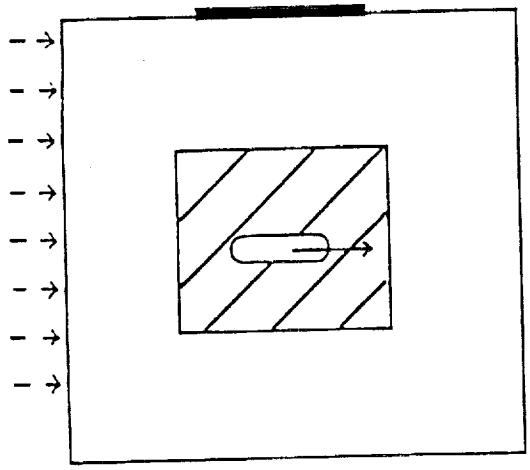
Fixed Region of Influence



CLASS II

Free Boundary Motion
(Size-Variable Domain)

Changing Region of Influence



CLASS III

Free Object Motion

Changing Region of Influence

KEY:

//// Region of Influence

⟶ Object or Boundary Motion

- - -→ Fluid Influx

Figure 4.3: Samples of the three general problem classes.

2. Define a region of influence in the fluid domain through which the structure will be allowed to move.

3. Solve the structural problem and determine the nodal velocities for points on the structural boundary.

4. Initialize the smoothing iteration counter to zero.

5. Determine a stable time step for the fluid domain.

6. Check the computational fluid grid for tangling at the end of the time step using the nodal grid velocities from step 3.

7. If tangling occurs then:

   (a) if the smoothing iteration counter is less than a given maximum smooth the grid inside the influence region using a node relocation technique and interpolate for new nodal values. Go to step 6.

   (b) if the smoothing iteration counter is greater than a given maximum remesh inside the influence region (possibly with triangular elements) and interpolate for new nodal values. Go to step 4.

8. Solve the fluids problem with the specified boundary motion.

9. Update the nodal coordinates of the fluid grid.

10. Smooth the fluid grid if orthogonality or smoothness is violated.

11. Check for convergence in the fluid region.

12. If convergence is achieved STOP.

13. Otherwise, check and see if redefinition of the influence region is needed. If so, determine a new region of influence and project the current grid points onto the background grid.

14. Go to step 3.

Presently all steps except 7(b) and 13 have been implemented into a pilot code for solving FSI problems with linearly elastic structural response.

To implement such a methodology a remeshing algorithm to relocate nodal points in the influence region is needed. The remeshing algorithm is based on a method presented by Carcaillet, Dulikravich, and Kennon [11]. A master element is defined as a block of four elements with a common node. Four vectors emanating from this node are delineated by the common sides of two adjacent elements in the master element. Using these vectors, an expression is derived that measures the regularity of the elements:

$$F(V) = \alpha(ORT) + (1 - \alpha)(SM) \tag{4.2}$$

where V represents the coordinates of the centeral node and α is a variable that emphasizes orthogonality vs. smoothness of the elements. The orthogonality function consists of the sum of the squares of the dot products of the consecutive vectors; the smoothness is measured by adding the squares of the differences of the areas of consecutive elements where the area is approximated by the cross product of the bounding vectors. The functional (4.2) is applied to all master element nodes in the region of influence. A function $\psi(\omega_n)$ is defined as $F(V^{n+1})$, where

$$V^{n+1} = V^n + \omega_n \delta V \tag{4.3}$$

and ω_n is a line search parameter. The function ψ is minimized with respect to the line search parameter and the new location of V is then calculated. Figure 4.4 presents a visual description of a master element and an outline of the remeshing algorithm. The actual FORTRAN coding of the algorithm is presented in Appendix A.

(1) Define a master element about a point $V(x,y)$ consisting of the elements bordered by $\vec{r}_i$ $(i = 1,4)$ as shown in Figure A.

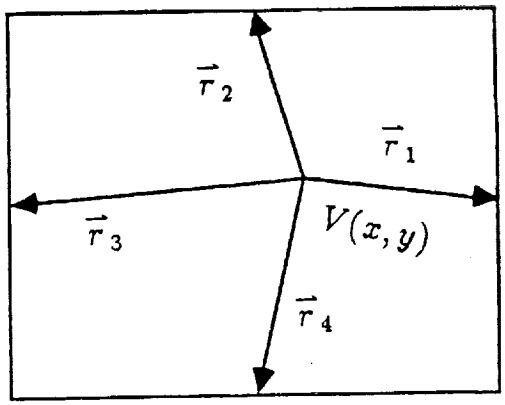


Figure A

(2) Calculate $F(V) = \alpha(ORT) + (1-\alpha)(SM)$ where

$$SM = (A_1 - A_2)^2 + (A_2 - A_3)^2 + (A_3 - A_4)^2 + (A_4 - A_1)^2$$
$$A_1 = \|(\vec{r}_1 \times \vec{r}_2)\|$$
$$ORT = (\vec{r}_1 \cdot \vec{r}_2)^2 + (\vec{r}_2 \cdot \vec{r}_3)^2 + (\vec{r}_3 \cdot \vec{r}_4)^2 + (\vec{r}_4 \cdot \vec{r}_1)^2$$
$$\alpha = \text{variable to emphasize orthogonality vs. smoothness } (0 \leq \alpha \leq 1)$$

(3) Let $V^{n+1} = V^n + \omega_n \delta V^n$ where

$$\delta V^\circ = -\nabla F(V^\circ)$$
$$\delta V^n = -\nabla F(V^n) + B_n \delta V^{n-1}$$
$$\beta_n = \|\nabla F(V^n)\|^2 / \|\nabla F(V^{n-1})\|^2$$

(4) Minimize $\psi(\omega_n) = F(V^{n+1})$ with respect to ω_n, a line search parameter.

(5) Solve for V^{n+1}.

Figure 4.4: Outline of the node relocation method for the local remeshing algorithm.

5 BENCHMARK PROBLEMS

The two adaptive approaches outlined in Section 4 have been used in the solution of a pair of benchmark problems. These test cases include a flexible wall problem with flow normal to the surface of the wall, and a rigid cylinder on a flexible shaft with flow parallel to the axis of the shaft. The initial configurations and grids for these two problems were described in Section 4 and are shown in Figs. 4.1 and 4.2. For these geometries, the Navier-Stokes equations have been solved with all open boundaries specified as subsonic inflow and all solid boundaries specified as no slip. Motion of the no slip deformable boundaries has been implemented as described in the previous sections.

5.1 Flexible Wall Problem

The first test case modeled is the subsonic flow of a compressible viscous fluid past a flexible wall. The computational region shown in Fig. 4.1 is composed of three distinct boundaries: a subsonic inflow boundary on the top, left, and right, a rigid no slip bondary on the bottom, and a flexible no slip boundary modeling the wall. (Here all subsonic inflow boundaries are considered to be open boundaries and, once initialized, the code determines whether the boundaries are inflow or outflow regions.) Initially, the inflow stream entering the computational region is specified as having a uniform streamwise velocity profile and a zero transverse velocity. Interior nodal points have prescribed initial values coinciding with the inflow conditions.

The calculations for this geometry have been performed using a Reynolds number of 3.14×10^5, a Prandtl number of 0.70, a Mach number of 0.45, and a free stream temperature of 530 degrees Rankin. Other problem parameters include:

$$\begin{aligned} \text{Specific heat ratio} &= 1.4 \\ \text{CFL number} &= 0.5 \\ \text{Elastic modulus} &= 5 \times 10^3 \\ \text{Poisson's ratio} &= 0.30 \end{aligned}$$

A Lapidus artificial dissipation mechanism was used to dampen oscillations near shocks and the Lapidus constant was set to 1.0.

5.1.1 Quasi-Steady-State Method

The flexible wall problem described above was initially solved using the grid shown in Fig. 4.1. For this configuration a steady–state solution was obtained (with a large convergence tolerance) as shown in Figs. 5.1(a)–(d). These figures show the density, pressure, and Mach contours, and the velocity vectors for the undeformed wall. The formation of a vortex at the

top downstream corner of the wall and a shock in the upstream region are clearly visible in these figures.

From this steady–state solution the pressure and viscous stress distributions along the wall were extracted. With these values a new deformed configuration of the wall was determined and the grid generation scheme was used to create a new fluid grid, Fig. 5.2(a). Using this grid a second steady–state solution was obtained, see Figs. 5.2(b)–(e). These figures exhibit a distinctly different character from the original solution: the upstream shock has disappeared and a strong vortex has developed behind the wall.

The same procedure was used repeatedly to determine subsequent motions of the wall until a steady–state deformation (less than three percent change in the structural deformation) was reached. Results of the second, third and fourth grid motions are shown in Figs. 5.3 to 5.5, respectively. Comparing these results, one sees a similar trend in all of the contour and velocity vector plots, where there remains a strong vortex just downstream of the flexible wall and the upstream shock has disappeared.

5.1.2 Local Remeshing Method

The local remeshing method was also used to solve the flexible wall problem but with a somewhat simpler grid as shown in Fig. 5.6. For this case a radius of influence of 0.8 units was selected which specified a subregion of the grid for which node relocation was employed (see Fig. 5.7). Two additional parameters were needed for the remeshing algorithm: an orthogonality parameter α and a smoothing parameter $1 - \alpha$. These values were set to 0.25 and 0.75, respectively.

During the simulation, the mass of the structure was ignored and a simple elasticity solution for the wall was used to determine the motions. As a result of this simplification small oscillations of the structure occurred over a large number of time steps requiring an additional parameter to be specified to stabilize the motion. The parameter selected was a three percent tolerance on the structural motions, i.e., when the maximum displacement of the structure changed by less than three percent of its previous value, the nodal displacement increments for the time step were zeroed out. This requirement allowed the structural motion to finally settle to the steady–state configuration shown in Fig. 5.8.

An expanded view of the deformed configuration of the cantilevered flexible wall showing the computational grid used in the structural calculations is presented in Fig. 5.9. This grid was internally generated by the code using the boundary points from the fluid to generate the outline of the structure and Lo's method [13] for determining the interior points. For this configuration, the σ-y stress contours for the wall were also plotted as shown in Fig. 5.10. As expected, the maximum stress concentrations appear near the fixed support region.

To trace the development of the steady–state configuration, the density contours were initially plotted every twenty time steps for the first 1020 steps. These results, see Figs. 5.11 (a)–(m), show the evolution of the density contours and the oscillations of the structural motion. By following this evolution sequence, one finds a vortex developing at the upper downstream corner of the wall (140 steps) which eventually breaks free and moves to a

downstream position where it becomes stationary (step 620). At this time a second vortex begins to develop behind the wall as indicated in steps 800 to 1020. After a number of additional steps, this vortex directly behind the wall dissipates leaving a single downstream vortex as in the quasi-steady-state method.

The final steady state results for the density, pressure and velocity are shown in Figs. 5.12 to 5.14. These results compare favorably with those obtained from the quasi-steady-state method even though considerably different grids have been used in the modeling. It is apparent from both sets of results that a finer grid would have been desirable to reduce the oscillations in the contour plots. Time and computational resources, however, restricted the number of elements in each model.

5.2 Rigid Cylinder on a Flexible Shaft

The second test case modeled is the subsonic flow of a compressible viscous fluid past a rigid cylinder attached to a flexible shaft. The computational region shown in Fig. 4.2 is composed of a subsonic inflow boundary along the top, bottom, left, and right edges, and a no slip boundary representing the shaft and rigid cylinder. The computational domain has dimensions of $14R \times 14R$ (where R is the cylinder radius) and has been discretized with approximately 2500 elements.

The simulation was performed using a Reynolds number of 3.14×10^5, a Prandtl number of 0.70, and a free stream temperature of 530 degrees Rankin. The inflow conditions for the upper and lower sections of the outer boundary have been divided along the axis of the shaft ($y = 0$). For points above this axis the inflow Mach number is set to 0.30 while for points below the inflow Mach number is 0.45. Other parameters used in the computations are:

Specific heat ratio	=	1.4
CFL number	=	0.5
Elastic modulus	=	2.0×10^4 (shaft)
	=	2.0×10^8
Poisson's ratio	=	0.30

Again, Lapidus artificial dissipation has been employed with the constant set to 1.0.

5.2.1 Quasi-Steady-State Method

The results of applying the quasi-steady-state method to the rigid cylinder problem are shown in Figs. 5.15–5.19. In this simulation the shaft has been decomposed into two sections: a rigid section of length R beginning at the left inflow boundary and a flexible section connecting the rigid section to the cylinder. The first set of figures, 5.15 (a)–(e), correspond to the steady–state solution of the cylinder in its initial undeformed position. As anticipated, the

nonsymmetric inflow conditions have created a pressure imbalance with the higher pressure contours occurring on the underside of the cylinder (see Fig. 5.15(b)).

Following a set of steps similar to those described in the first sample problem, a deformed configuration for the shaft and cylinder was obtained as seen in Fig. 5.16 (a). Solving the fluid dynamics equations for this geometry, a second steady–state solution for the pressure distribution was obtained which was again used to obtain a new deformed structural model. Results for the second, third, and fourth iterations are shown in Figs. 5.16 to 5.19, respectively. Comparing these figures, it is notable that the location of the downstream vortices are highly dependent on the final deformed configuration of the cylinder. Even for small changes in the deformation, as from the last two iterations, this can have a significant effect on the location of the downstream vortices.

5.2.2 Local Remeshing Method

The local remeshing method was applied to the cylinder problem but with a somewhat different set of material conditions. In this modeling, the shaft was assumed to be entirely elastic with no rigid segment next to the inflow boundary and the cylinder was assumed to be composed of an elastic material with the same modulus as the shaft. These simplifications were due to the limitations present in the boundary conditions of the fluids code and the ability to specify only a single elastic constant for the structural modeling.

In this simulation, the region of influence was selected to be 3.0 units which identified a region for node relocation as shown in Fig. 5.20. The orthogonality and smoothness parameters were specified as 0.25 and 0.75, respectively, and the mass of the structure was neglected. As in the first example a convergence tolerance of three percent was imposed on the structural displacements.

During the simulation, the structure initially deformed with the flexible cylinder moving upward. After approximately 300 time steps, however, the node relocation algorithm began to detect negative Jacobians due to tangling of the fluids grid. Upon extracting the deformed fluids grid at the time step (see Fig. 5.21 (an expanded view of this region is shown in Fig. 5.22.)), it is apparent that some difficulties with the algorithm were occurring at the downstream edge of the cylinder where a number of the elements had been excessively sheared. The cause of this difficulty was in the selection of an orthogonality constant that was too small (0.25). Investigations into various choices of larger values of α are discussed in the following section.

Although a converged solution for this problem had not been obtained, the solution at 300 time steps has been plotted. Figure 5.23 shows the grid and the final deformed configuration for the cylinder and flexible shaft. The σ-y stress contours are shown in Fig. 5.24. The density, pressure, Mach contours, and velocity vectors at 300 steps are shown in Figs. 5.25 to 5.28.

The time history evolution of the density contours for the first 300 steps has also been plotted in Fig. 5.29. Examining this sequence one observes that the node relocation method has performed quite well for approximately the first 200 steps with a significant amount of

smoothing occurring throughout the remeshing region.

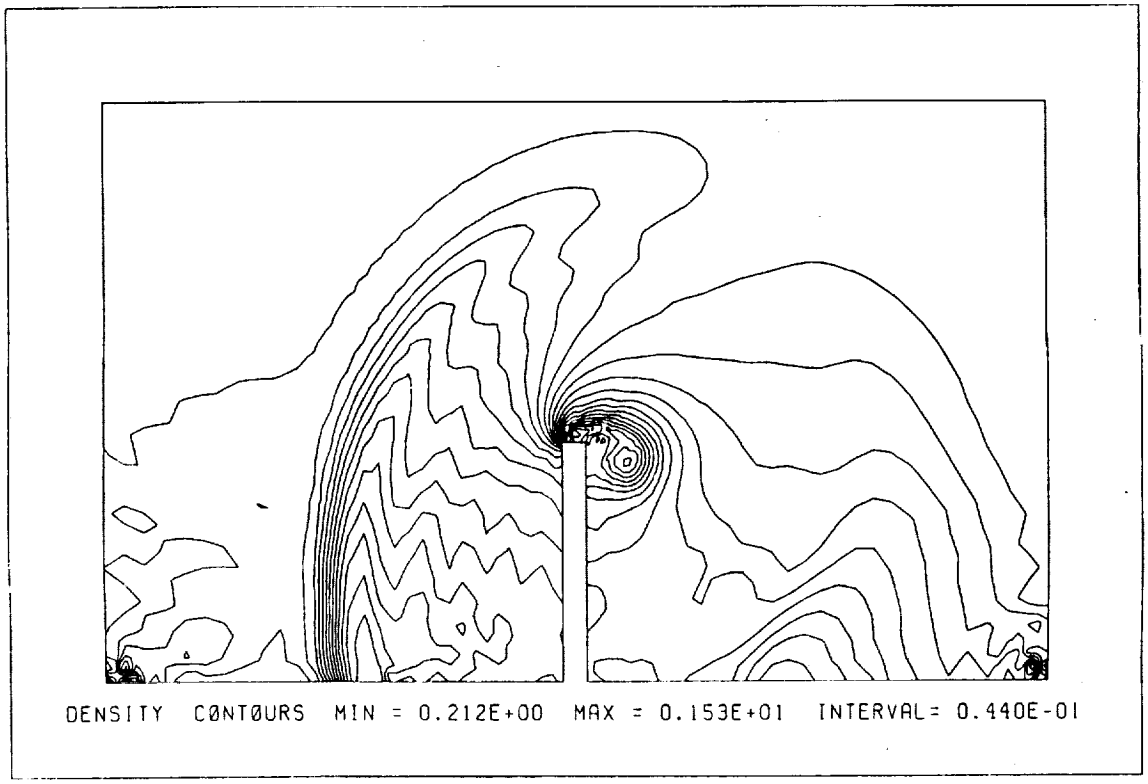
5.3 Remeshing Parameter Study

A study of the effects of changing the orthogonality and smoothness parameters was conducted in order to determine an optimal selection of parameters for the rigid cyinder/flexible shaft problem and to eliminate the mesh distortion near the cylinder. In this study fluid parameters and boundary conditions were the same as those used in Section 5.2.2. Four test cases were run; the choice of the smoothness/orthogonality parameters are listed in the table below.

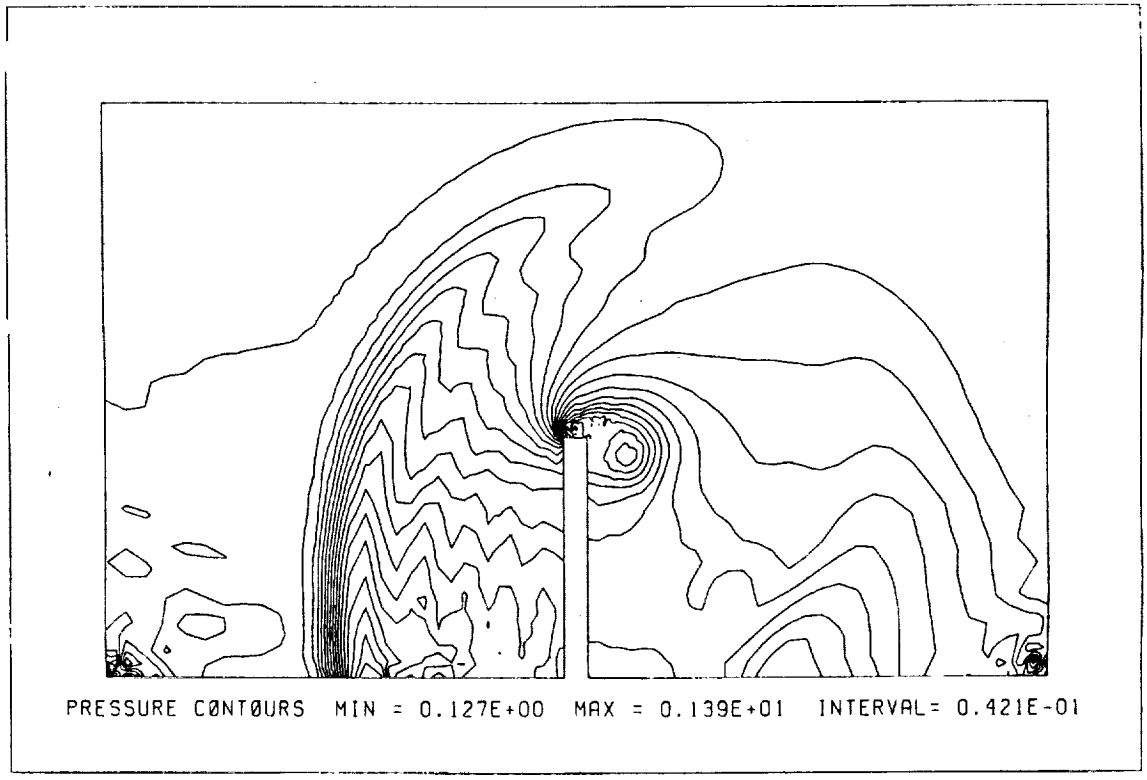
TABLE 1

Case	Orthogonality	Smoothness
1	0.25	0.75
2	0.40	0.60
3	0.60	0.20
4	0.80	0.20

Cases 2, 3, and 4 were each run for 1000 time steps; case 1 stopped execution after approximately 300 time steps. Results for these test cases are shown in Figs. 5.30–5.33. Enlarged views of the mesh around the cylinder show that mid–range values for the orthogonality parameter provide the best combination of smoothness and orthogonality while extreme values produce sheared grids. Taking the entire computational domain into account, the choice of an orthogonality constant of 0.40 appears to be the best choice (for those cases tested): the elements near the cylinder show little shearing and the majority of the domain shows a regularity in the sizes of the elements. Figure 5.34 shows the density contours for the case 2 problem extracted every 50 time steps.

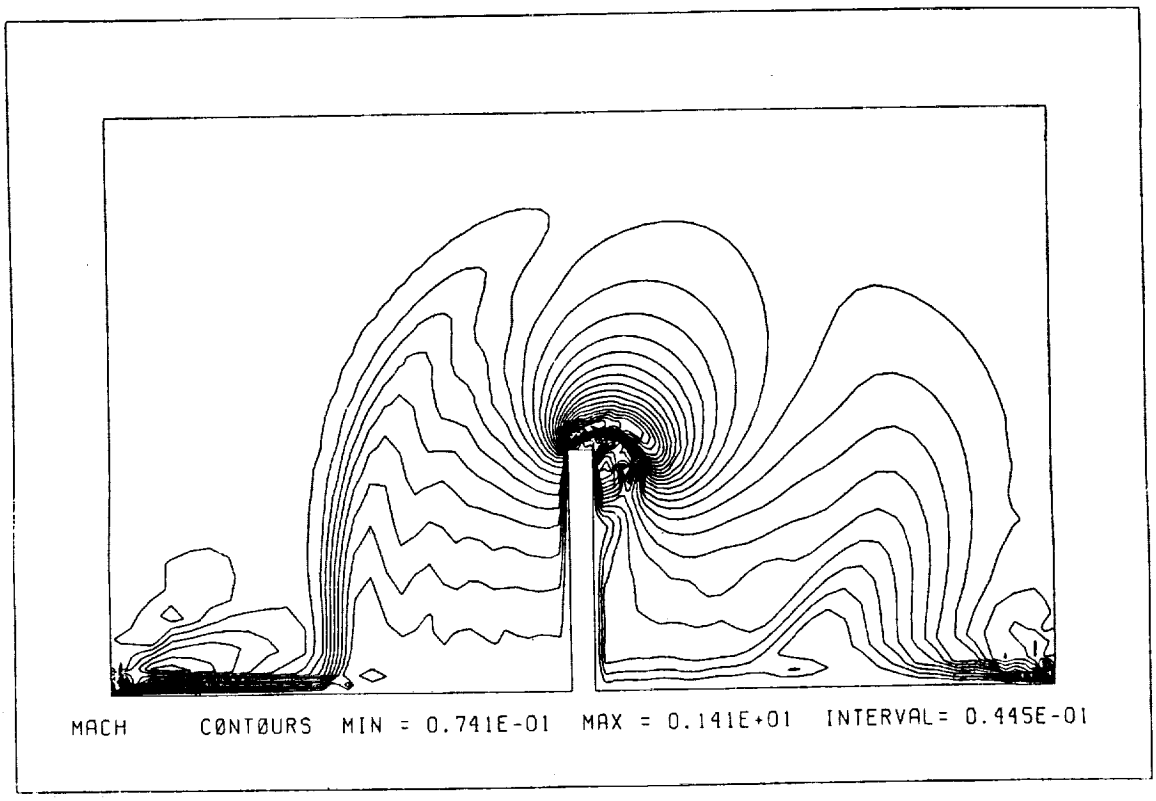


a

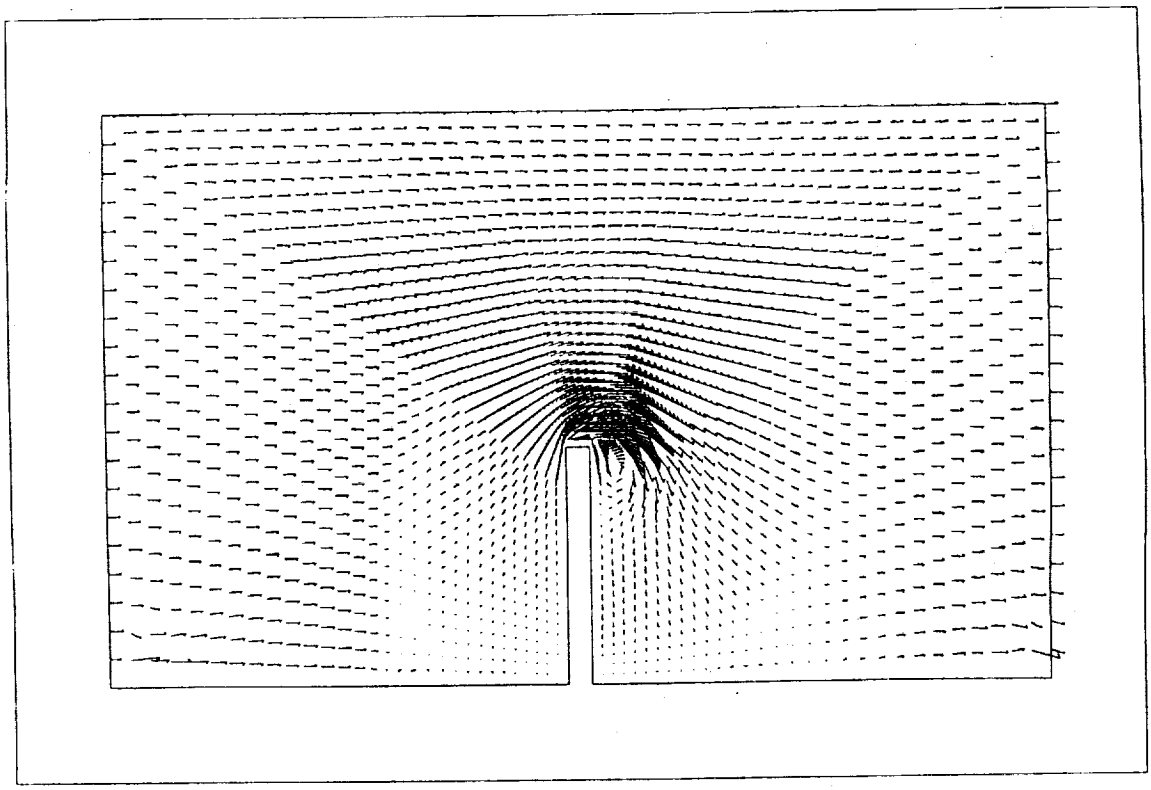


b

Figure 5.1a,b: Results for the flexible wall problem using the quasi–steady state method — nondeformed configuration. (a) Density contours, (b) pressure contours.



c



d

Figure 5.1c,d: Results for the flexible wall problem using the quasi-steady state method — nondeformed configuration. (c) Mach contours, (d) velocity vectors.

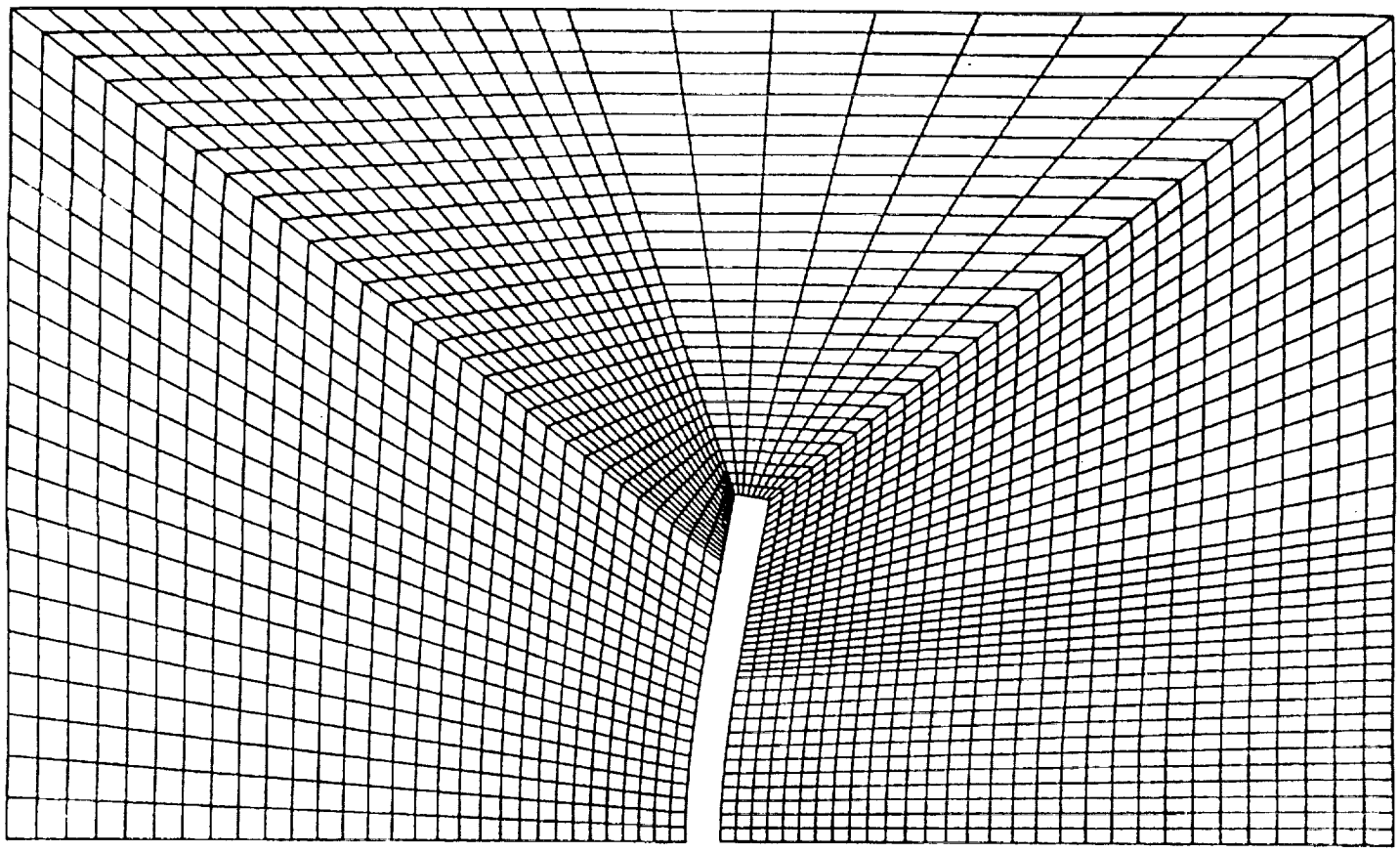
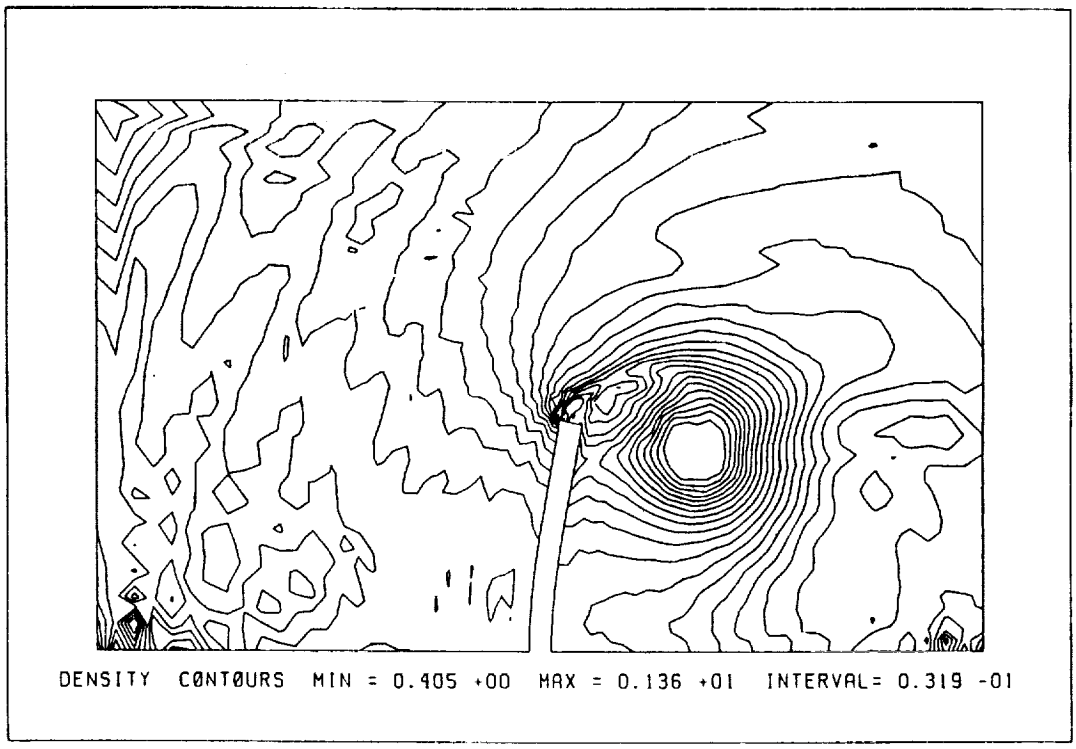
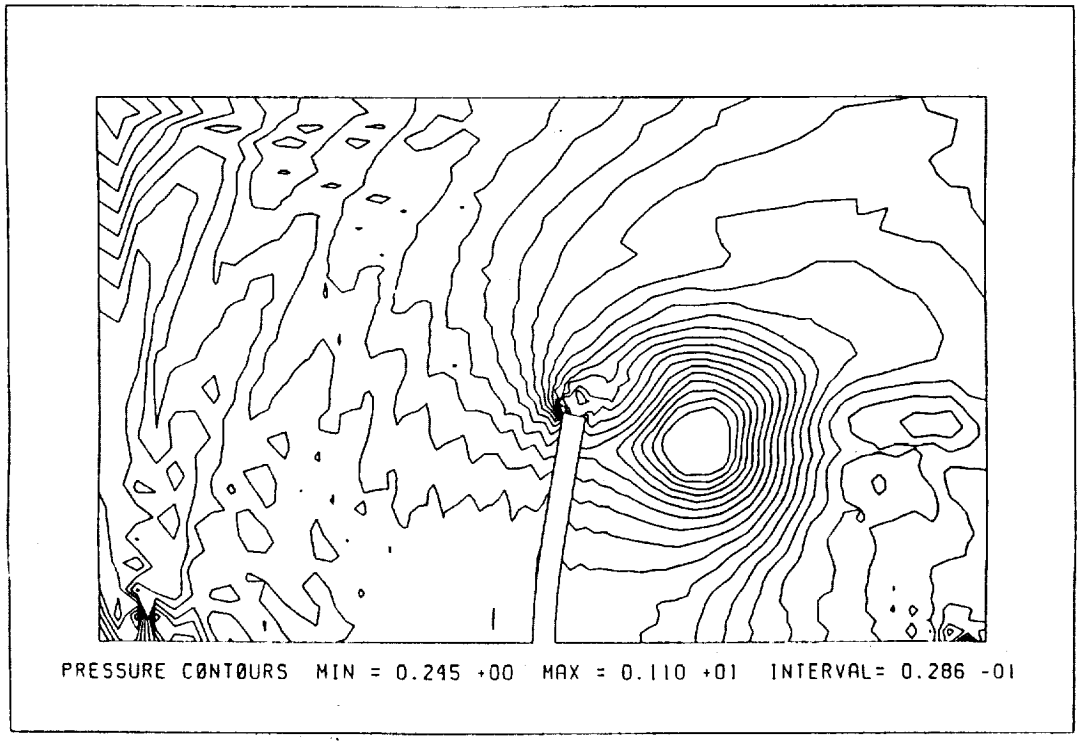


Figure 5.2a: First deformed configuration of the computational domain for the flexible wall problem using the quasi-steady state method.

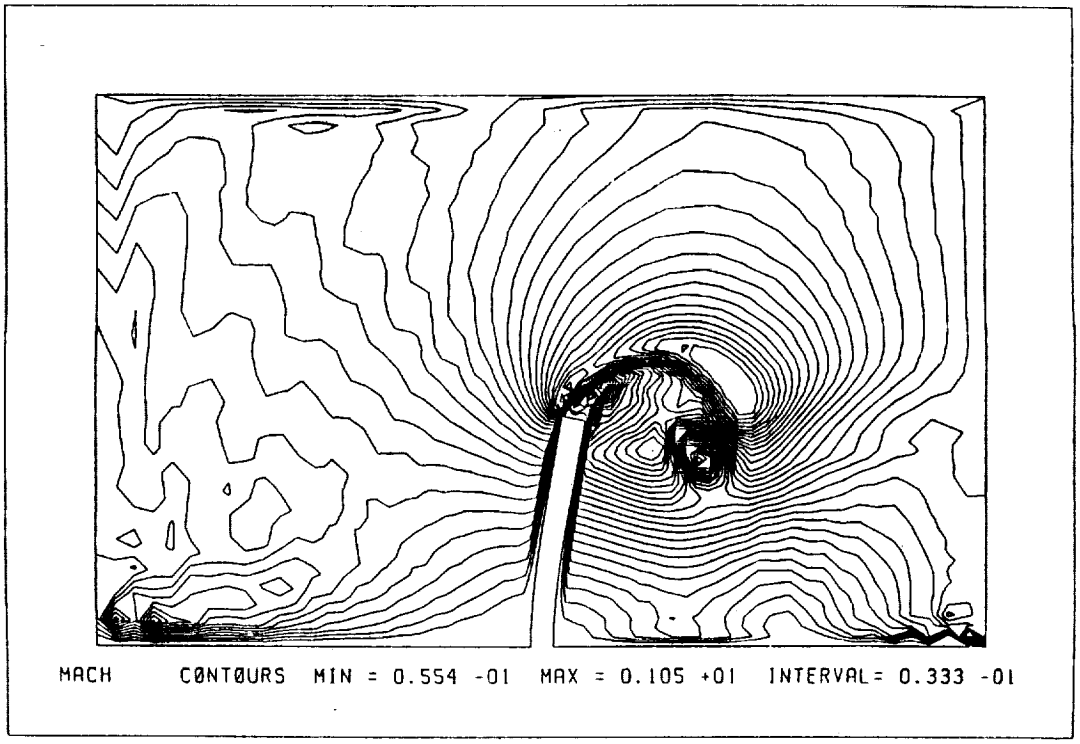


b

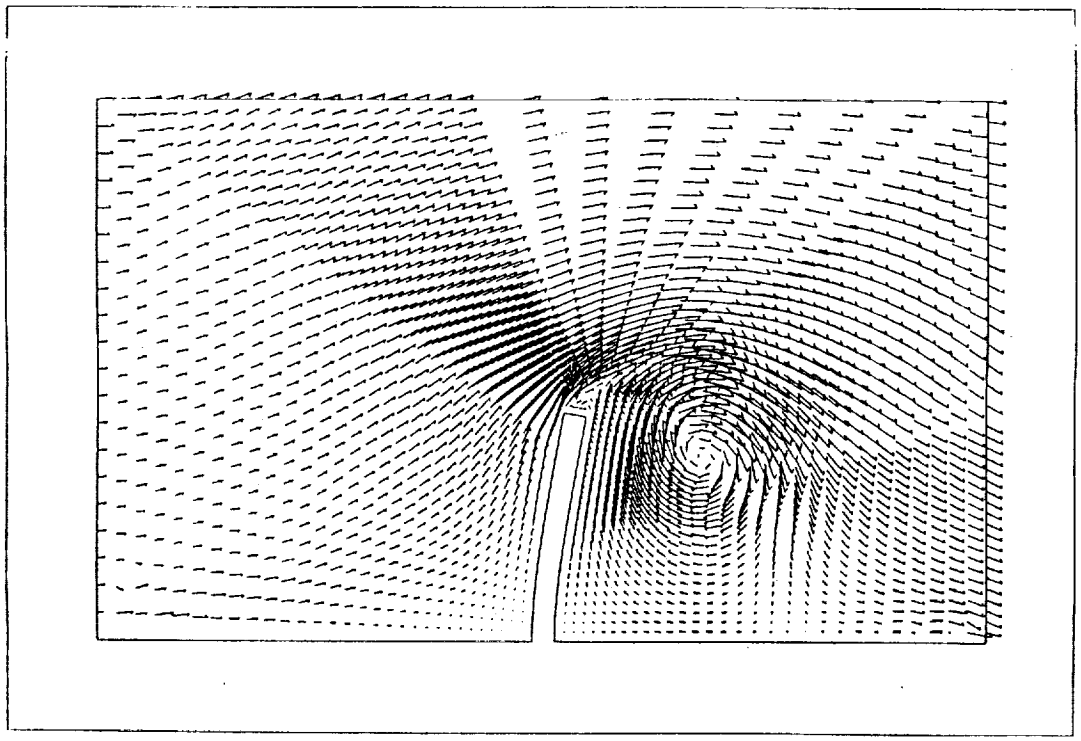


c

Figure 5.2b,c: Results for the flexible wall problem using the quasi–steady state method—first deformed configuration. (b) Density contours, (c) pressure contours.



d



e

Figure 5.2d,e: Results for the flexible wall problem using the quasi-steady state method—first deformed configuration. (d) Mach contours, (e) velocity vectors.

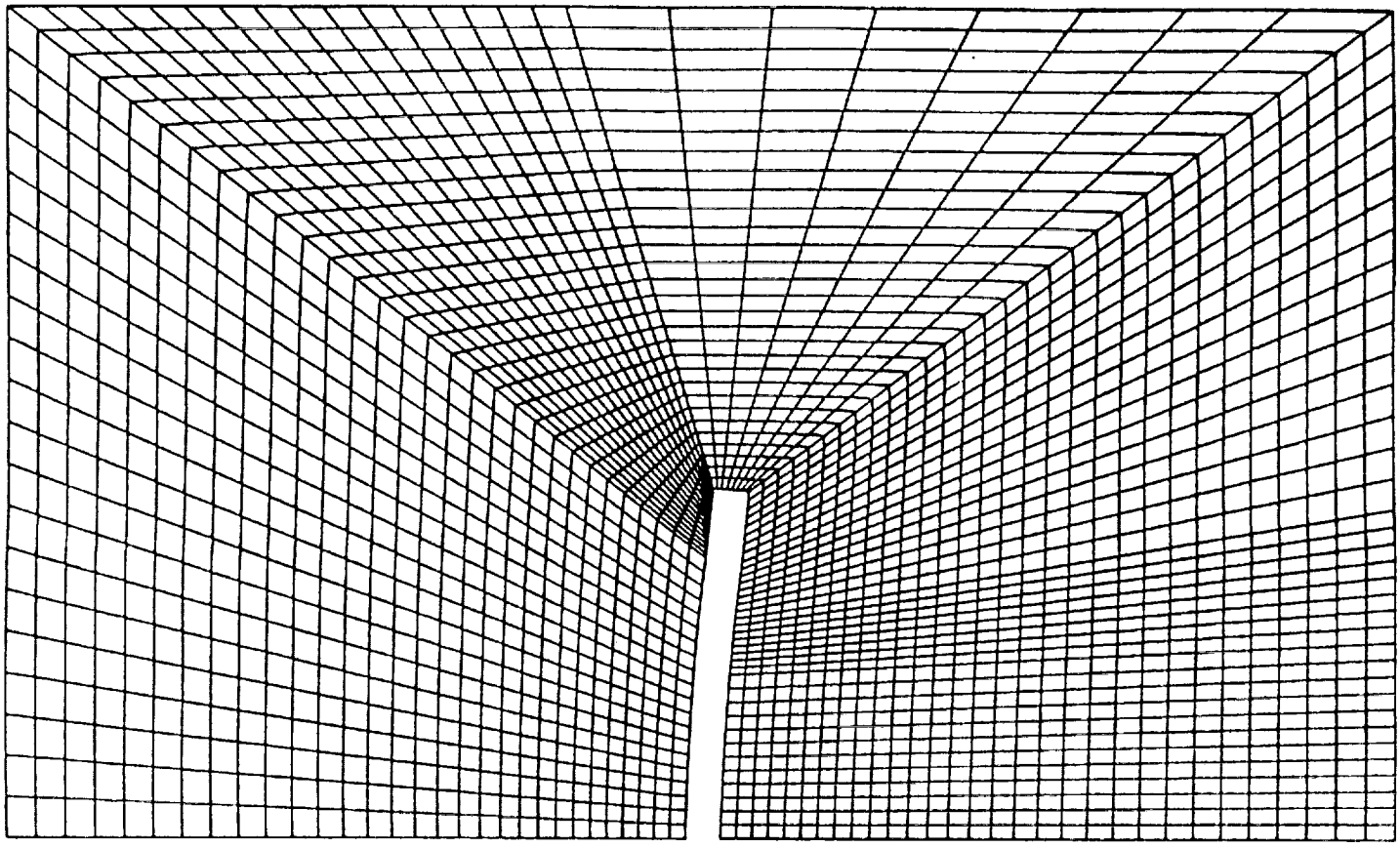
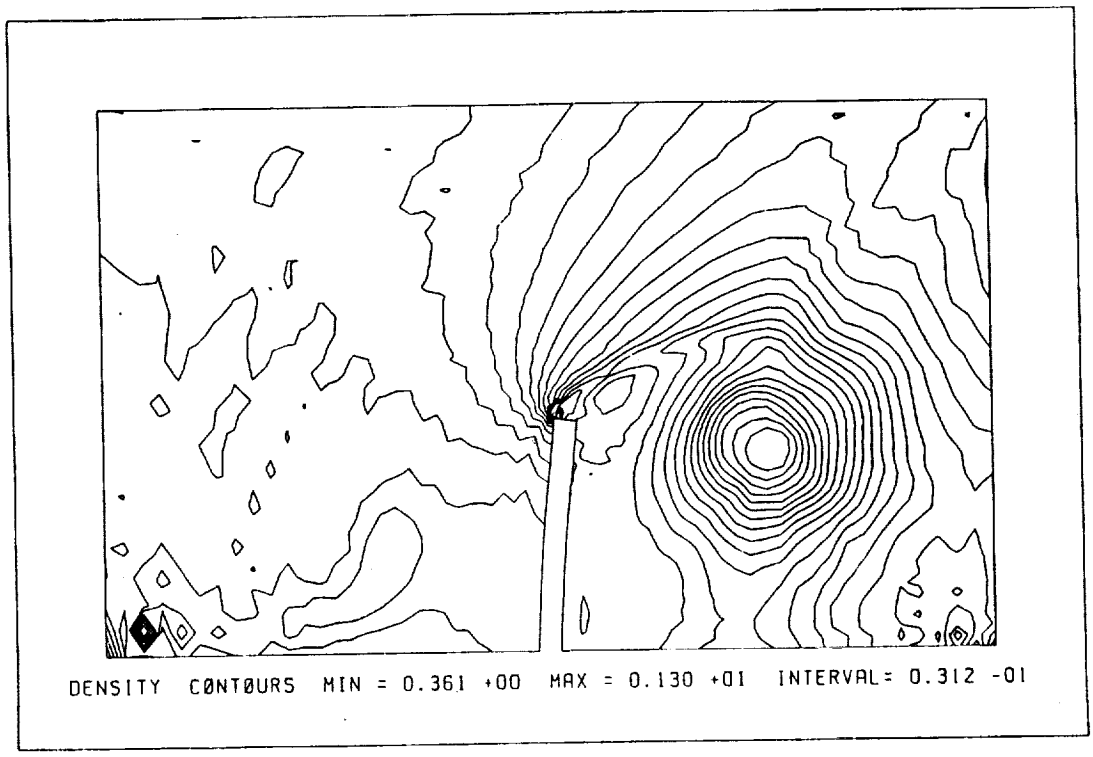
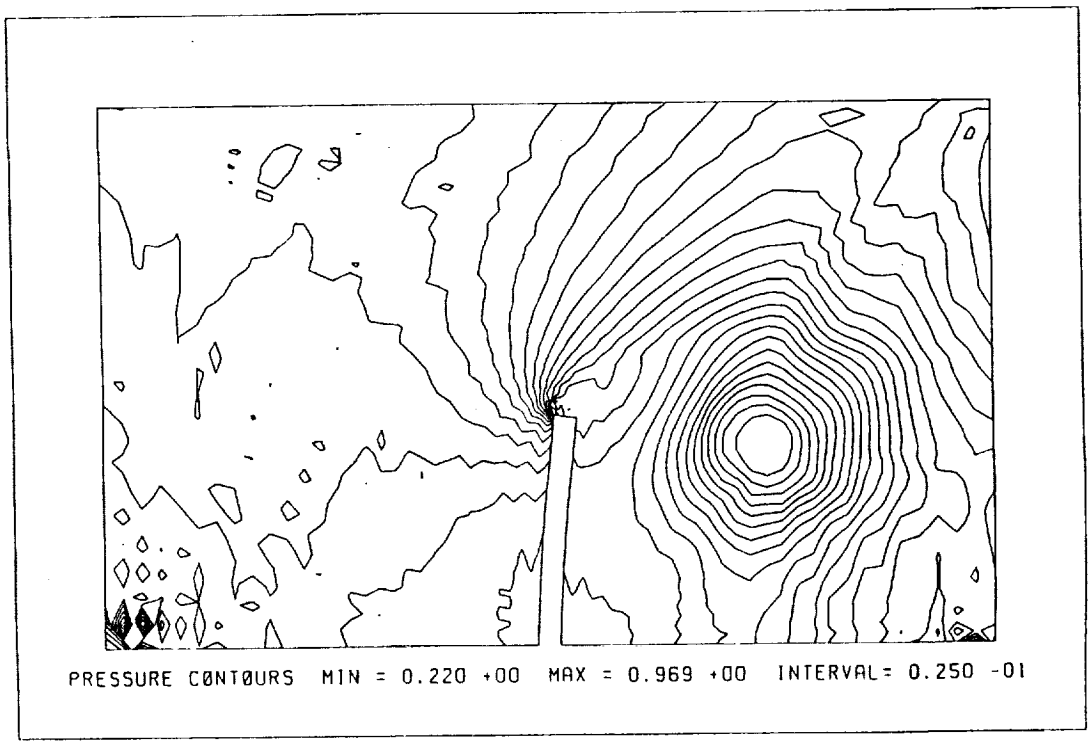


Figure 5.3a: Second deformed configuration of the computational domain for the flexible wall problem using the quasi-steady state method.

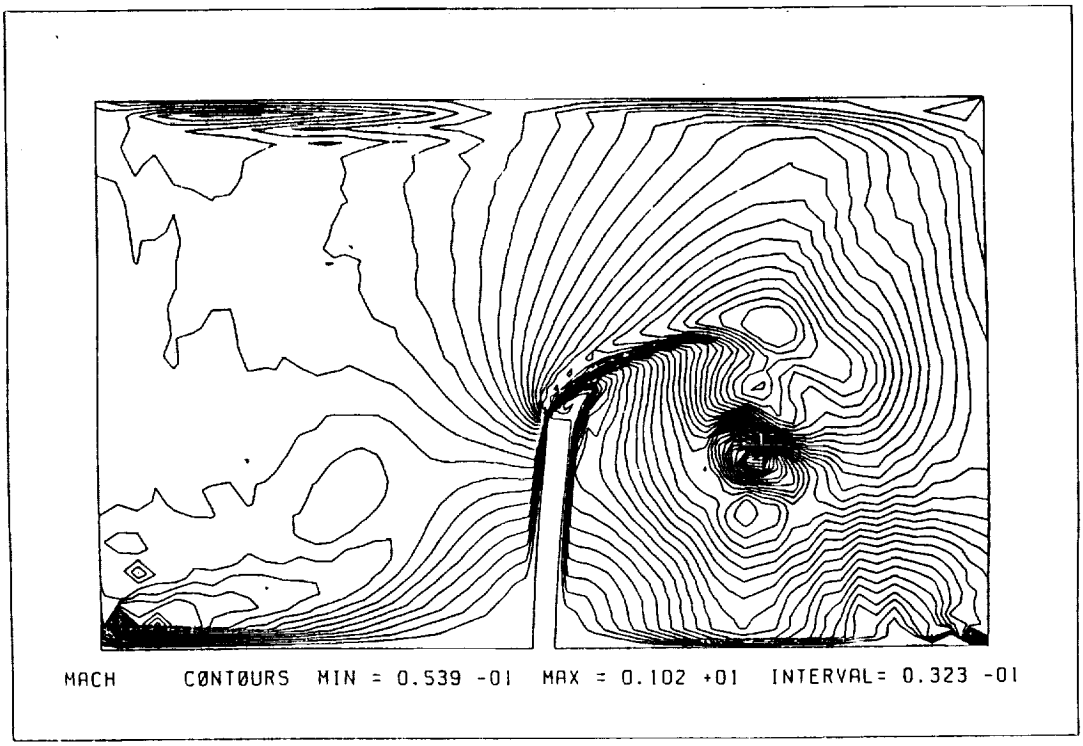


b

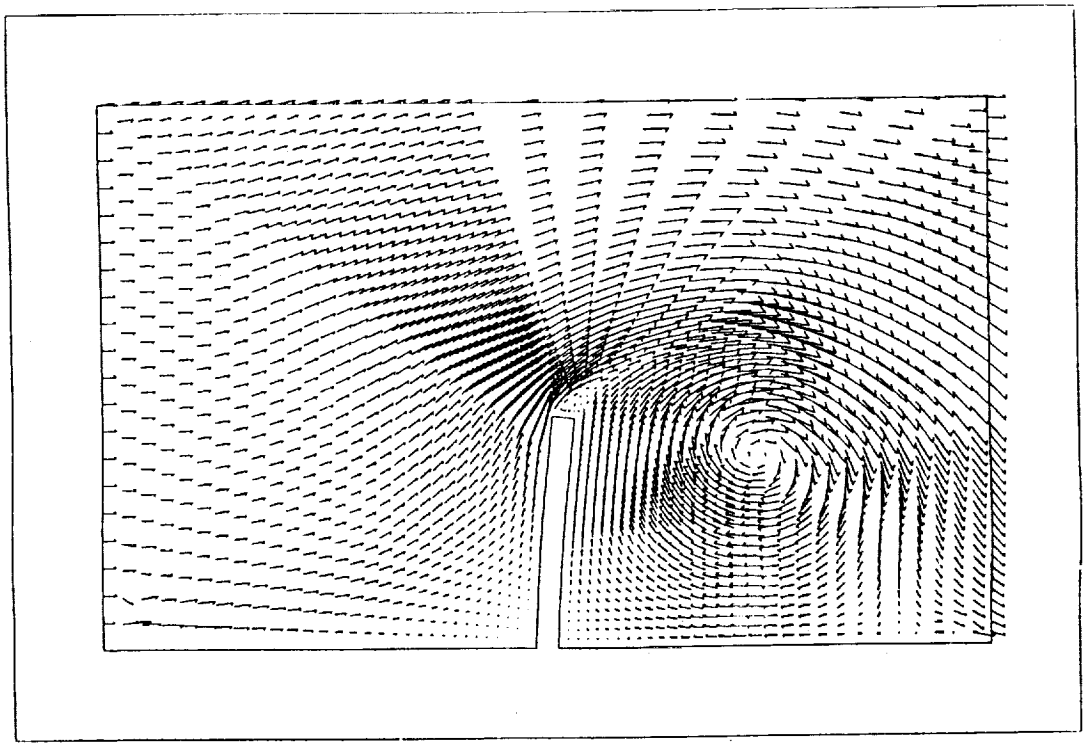


c

Figure 5.3b,c: Results for the flexible wall problem using the quasi–steady state method—second deformed configuration. (b) Density contours, (c) pressure contours.



d



e

Figure 5.3d,e: Results for the flexible wall problem using the quasi–steady state method—second deformed configuration. (d) Mach contours, (e) velocity vectors.

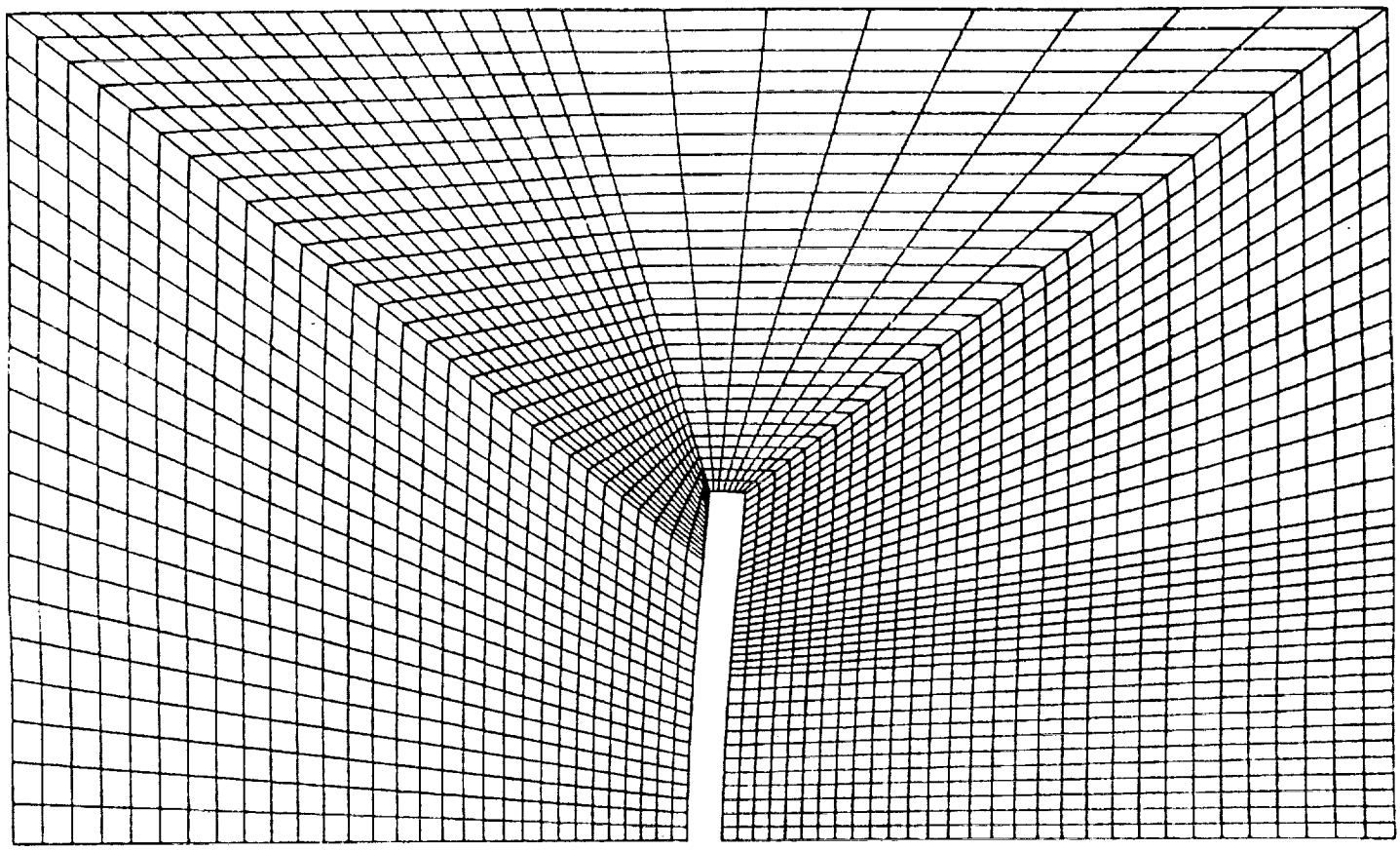
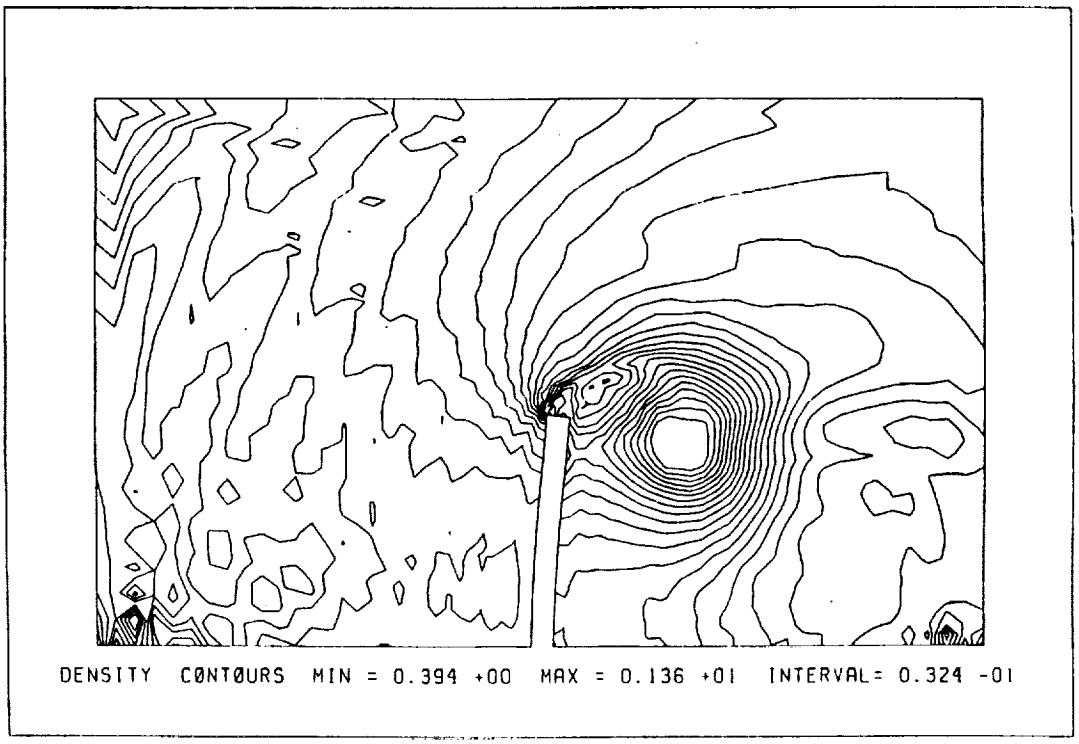
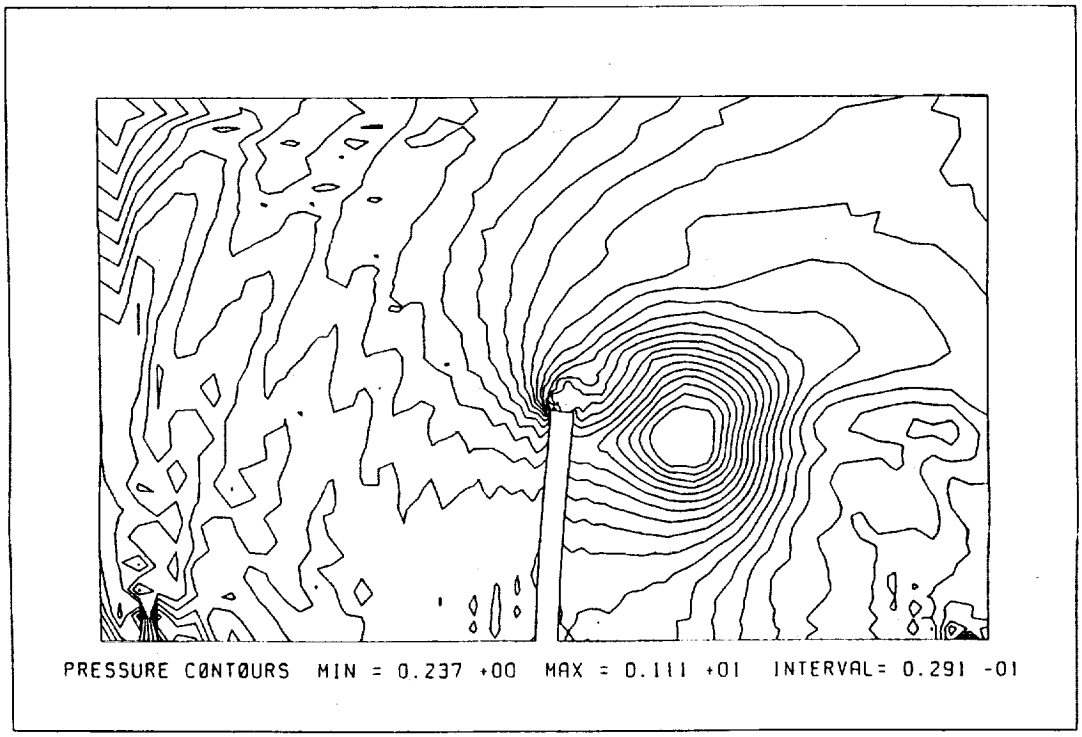


Figure 5.4a: Third deformed configuration of the computational domain for the flexible wall problem using the quasi–steady state method.

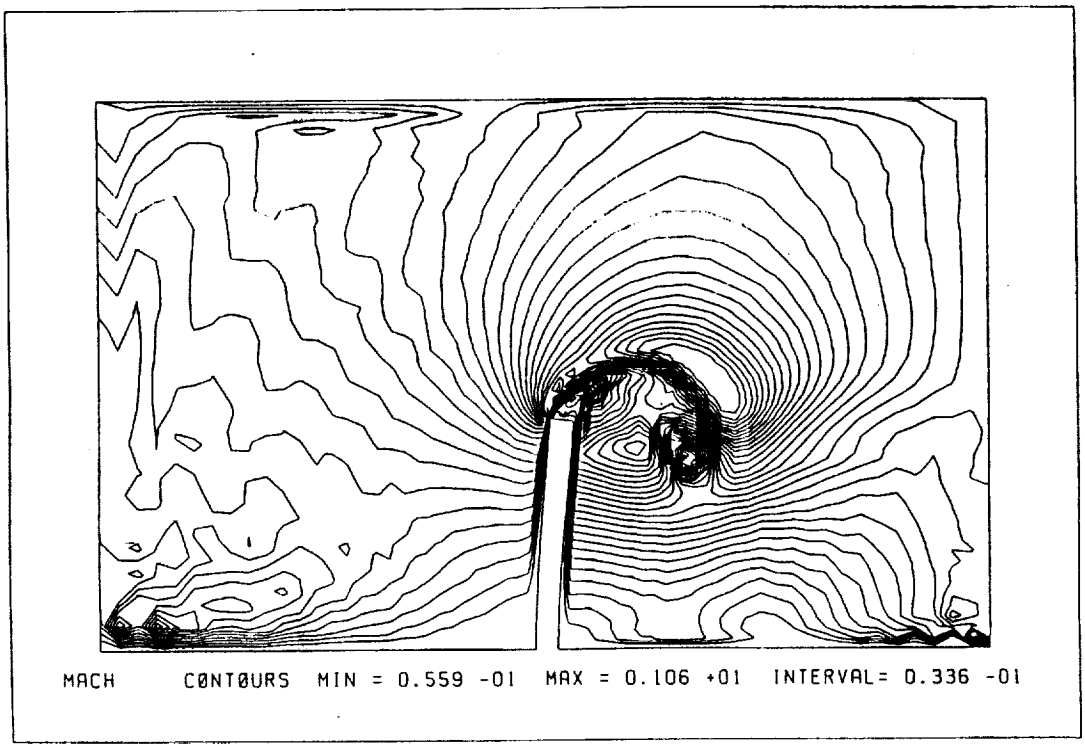


b

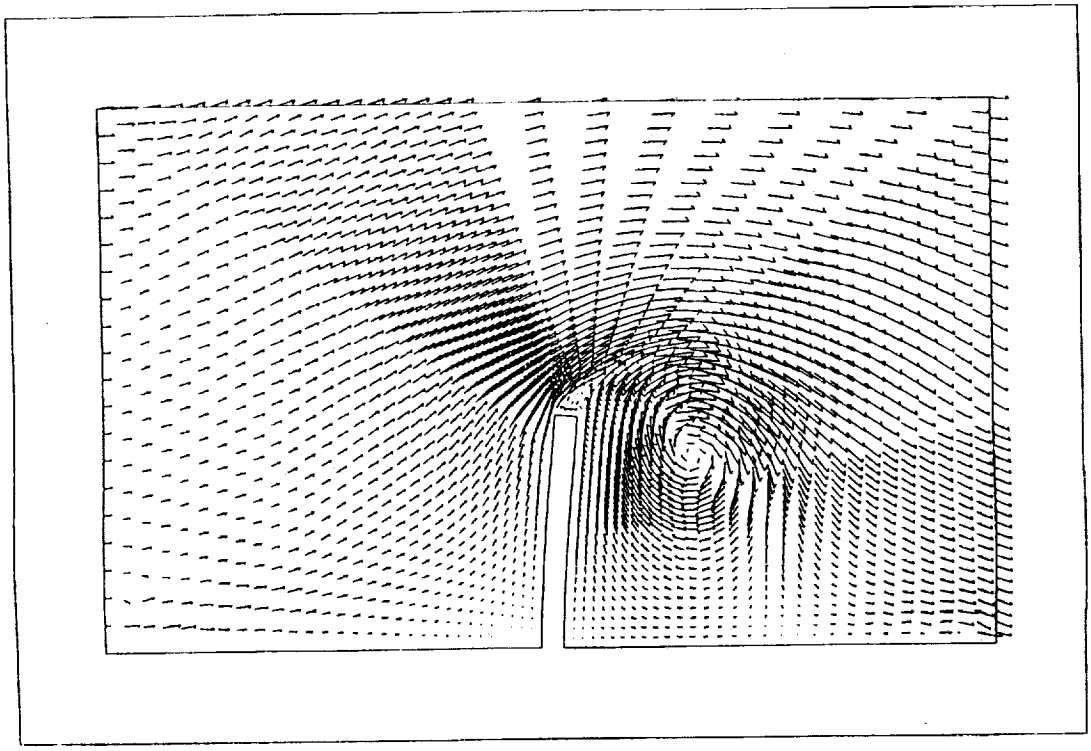


c

Figure 5.4b,c: Results for the flexible wall problem using the quasi-steady state method—third deformed configuration. (b) Density contours, (c) pressure contours.



d



e

Figure 5.4d,e: Results for the flexible wall problem using the quasi-steady state method—third deformed configuration. (d) Mach contours, (e) velocity vectors.

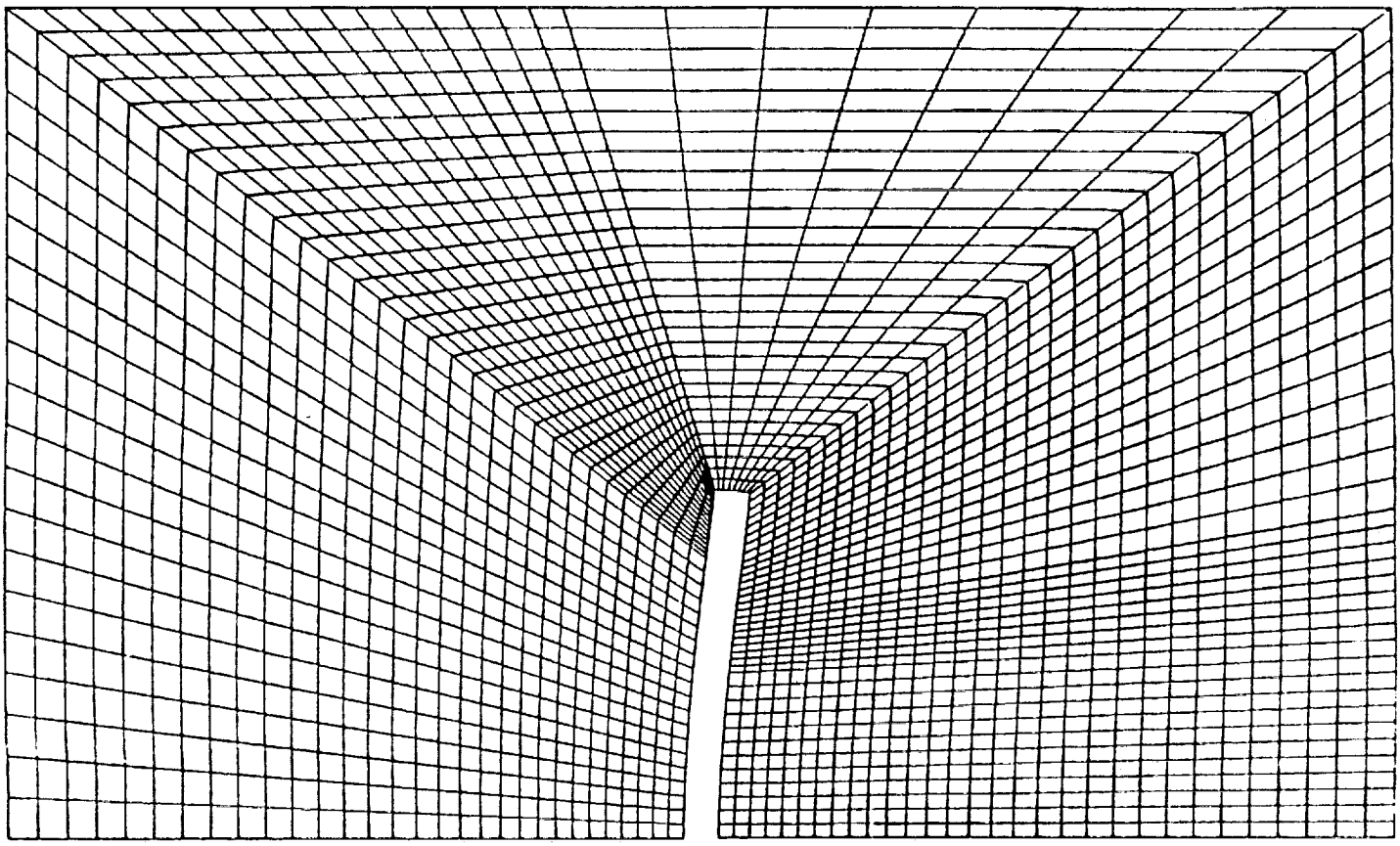
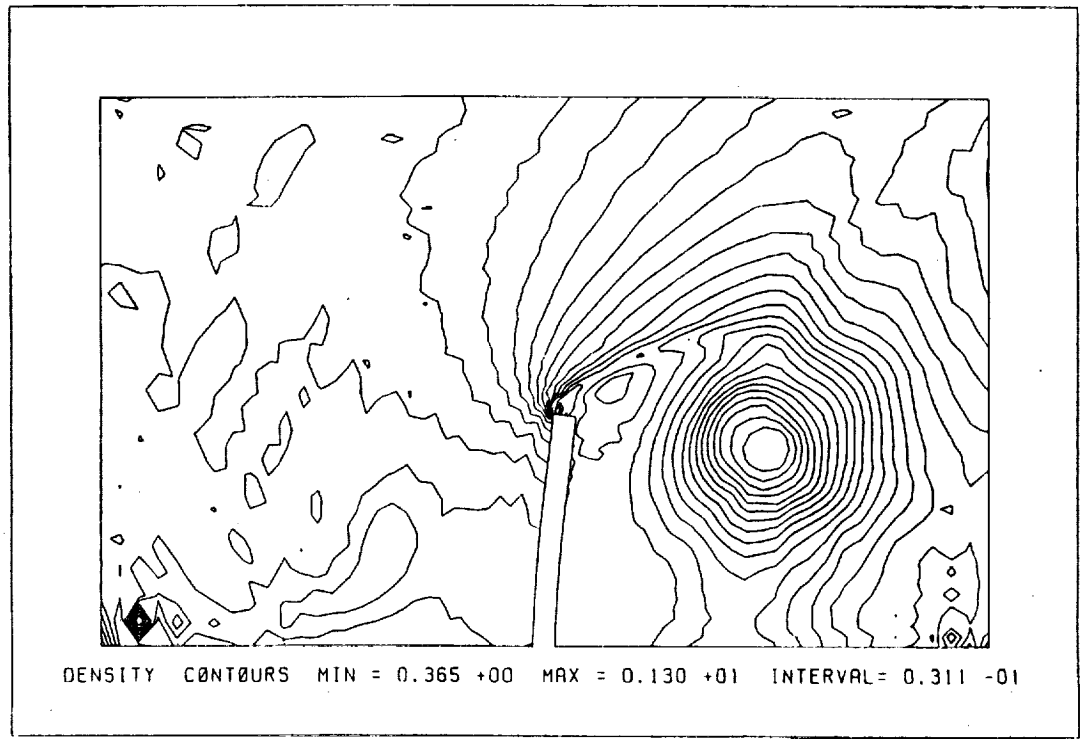
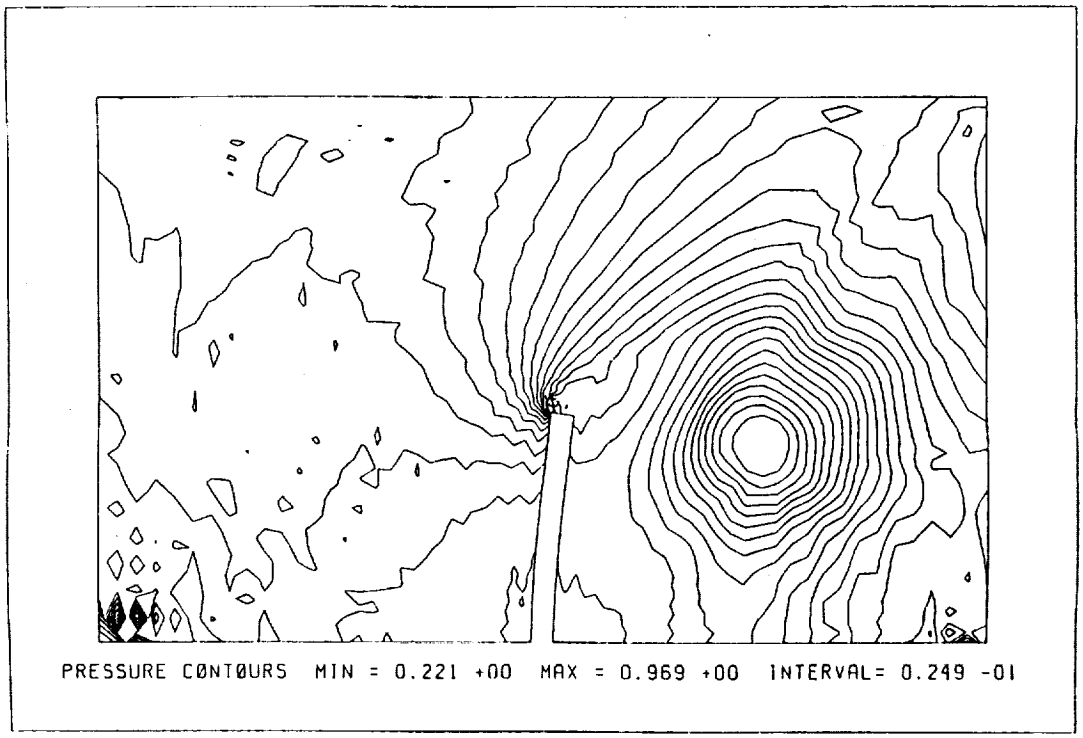


Figure 5.5a: Fourth deformed configuration of the computational domain for the flexible wall problem using the quasi-steady state method.

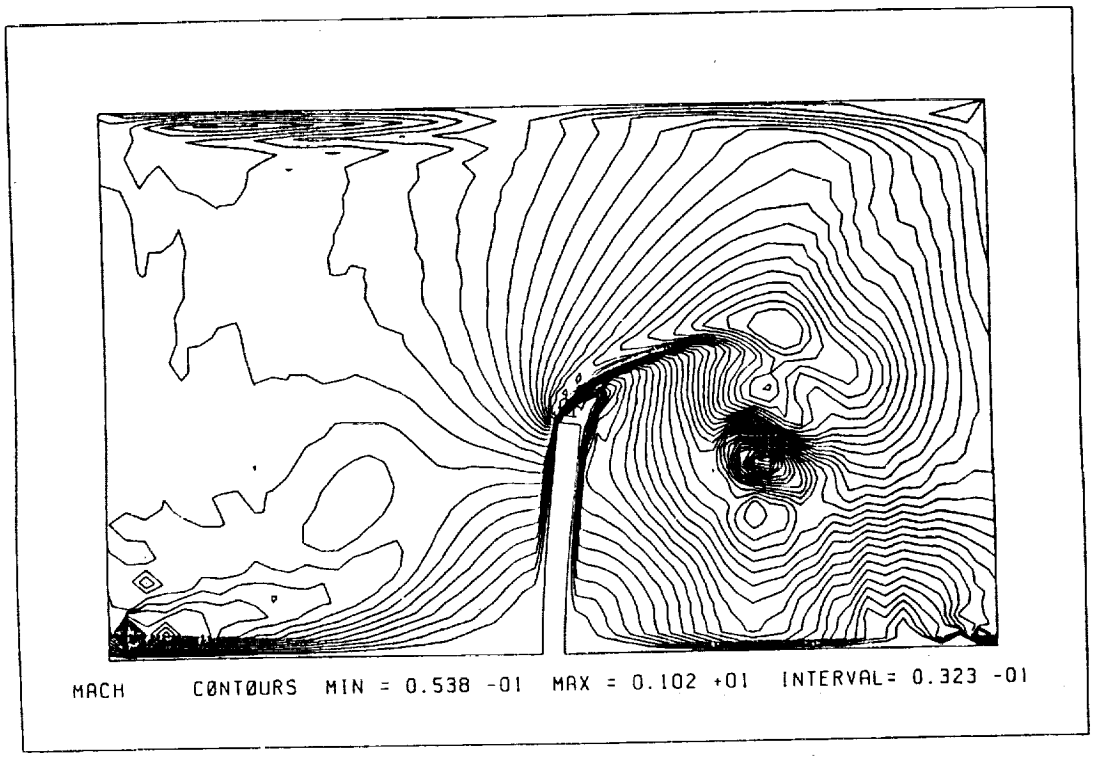


b

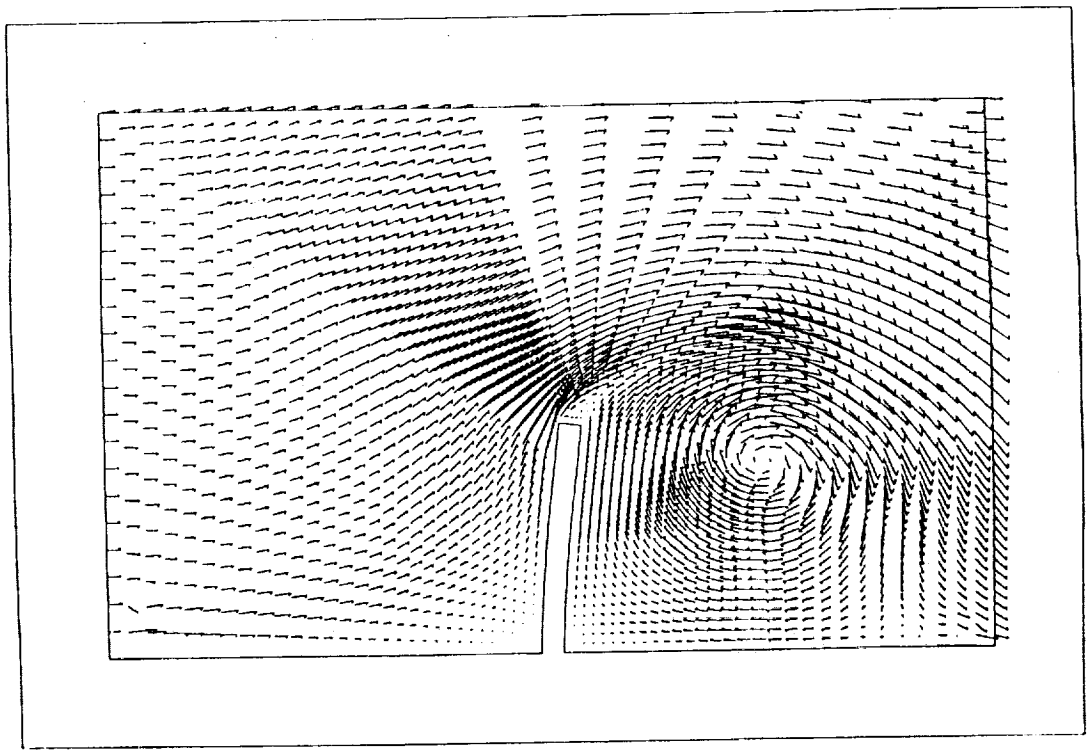


c

Figure 5.5b,c: Results for the flexible wall problem using the quasi–steady state method—fourth deformed configuration. (b) Density contours, (c) pressure contours.



d



e

Figure 5.5d,e: Results for the flexible wall problem using the quasi-steady state method—fourth deformed configuration. (d) Mach contours, (e) velocity vectors.

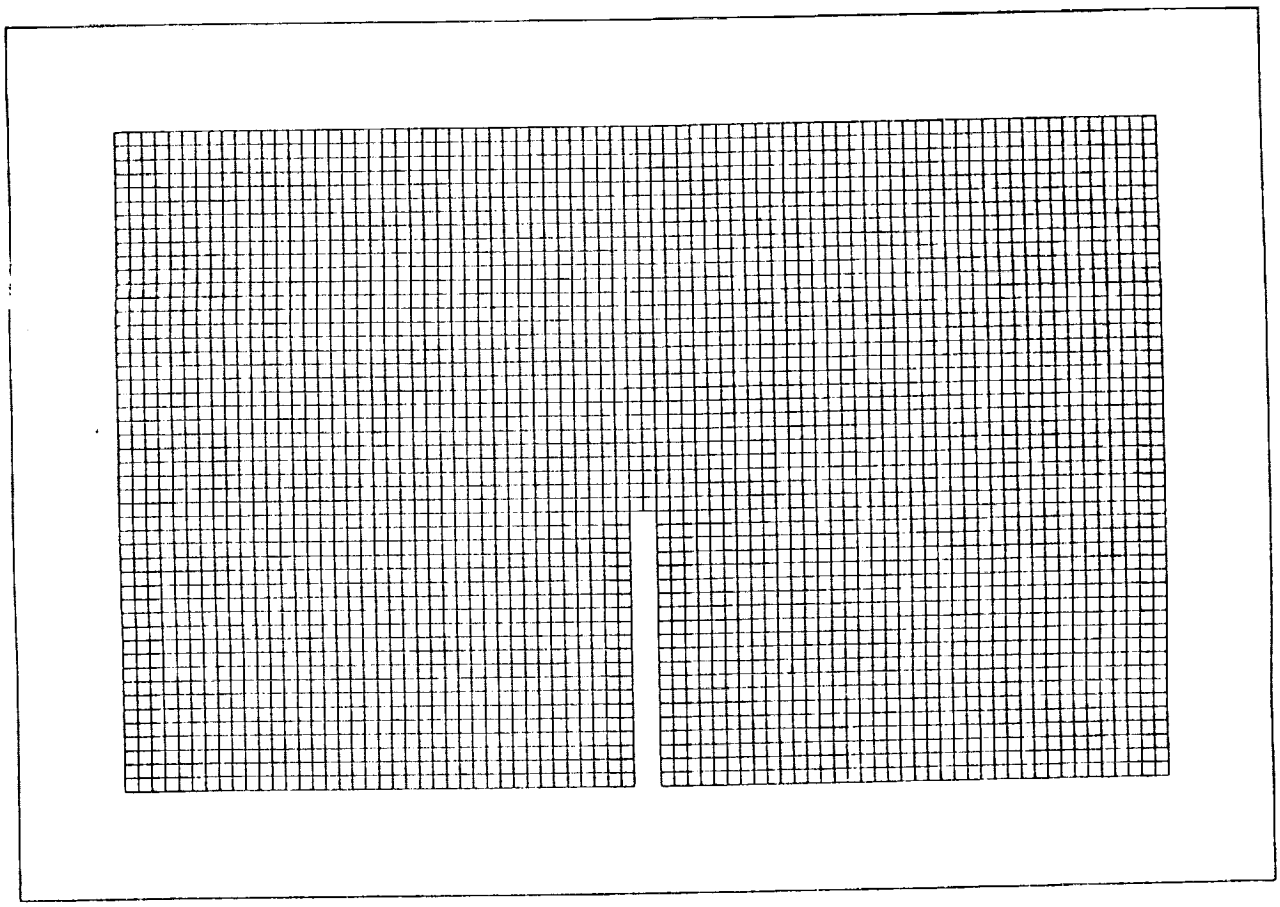


Figure 5.6: Initial uniform grid for the flexible wall problem using the local remeshing method.

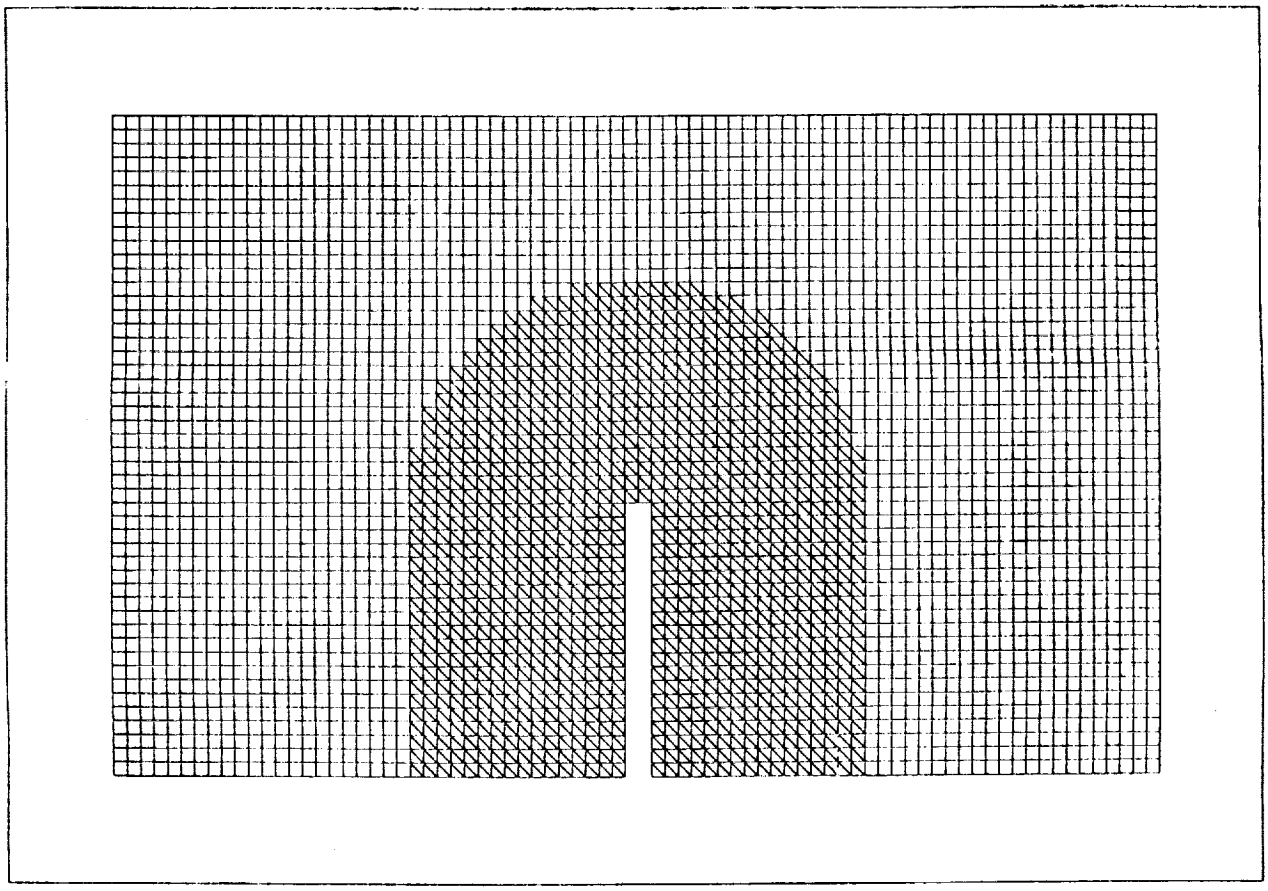


Figure 5.7: Initial uniform grid for the flexible wall problem with remeshing region cross-hatched.

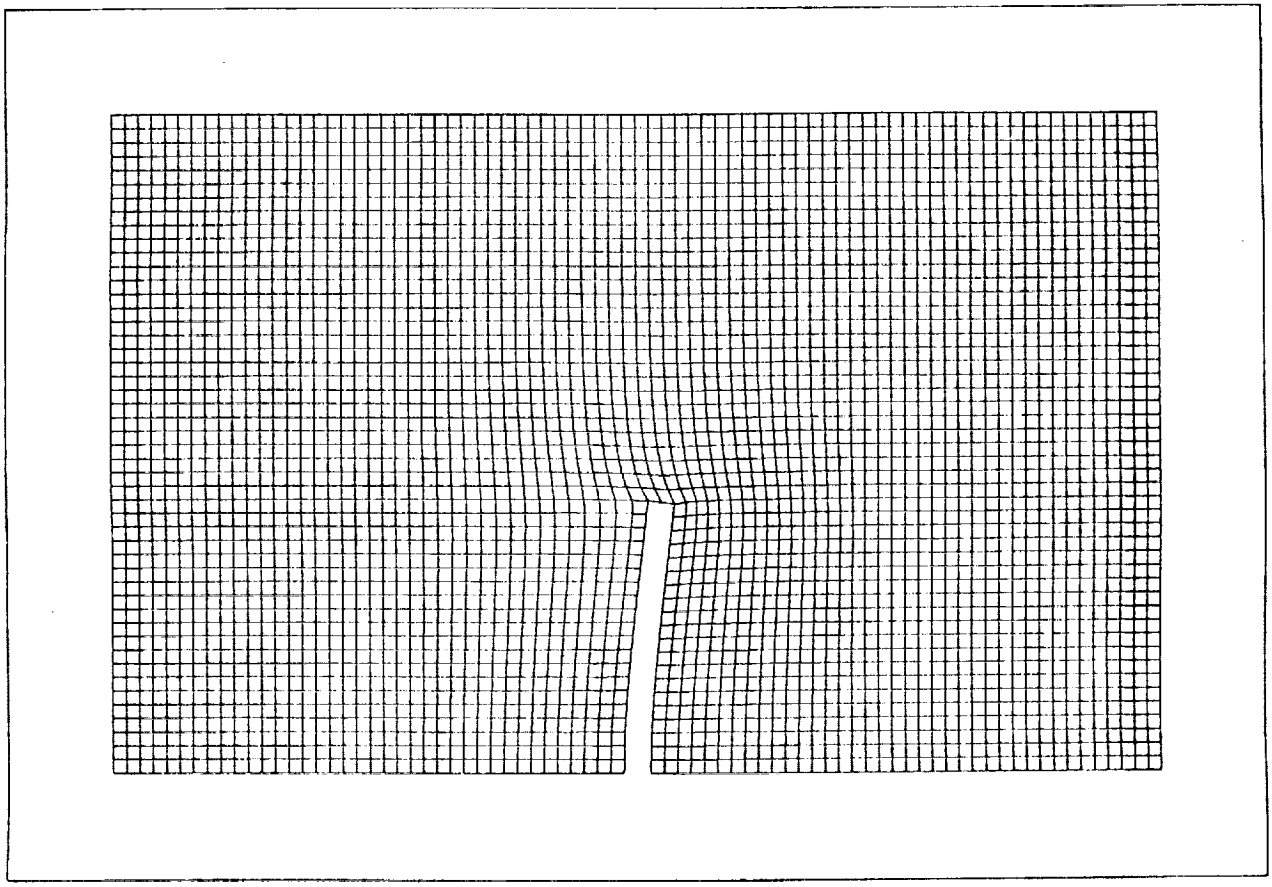


Figure 5.8: Final deformed grid for the flexible wall problem using the local remeshing method.

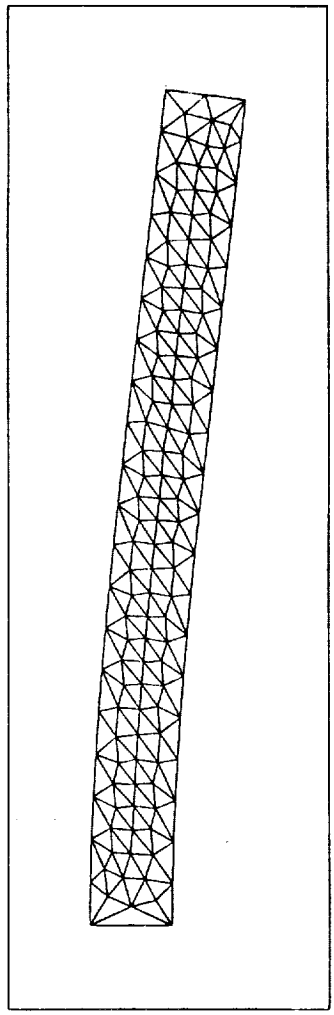


Figure 5.9: Deformed finite element grid used for structural modeling of the flexible wall with the local remeshing method. This grid was internally generated by the code after specifying the boundary points.

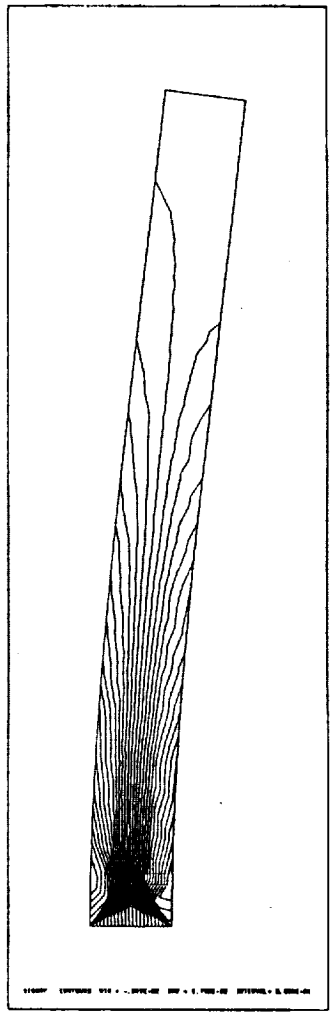
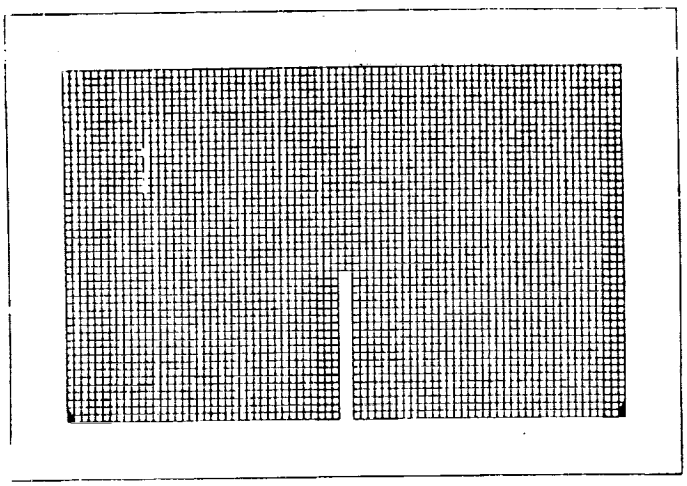
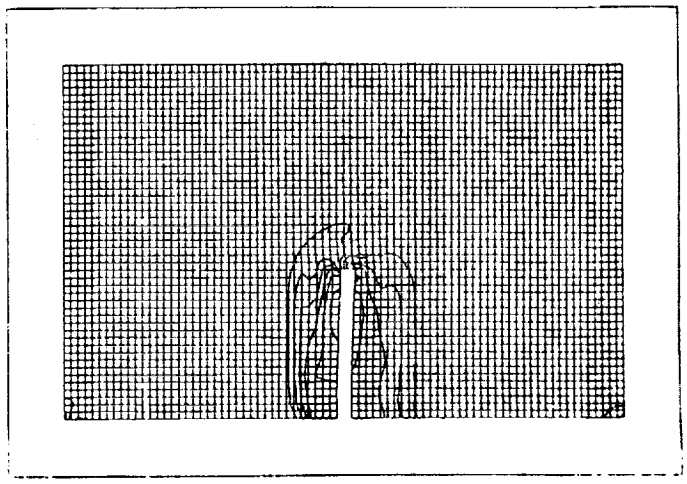


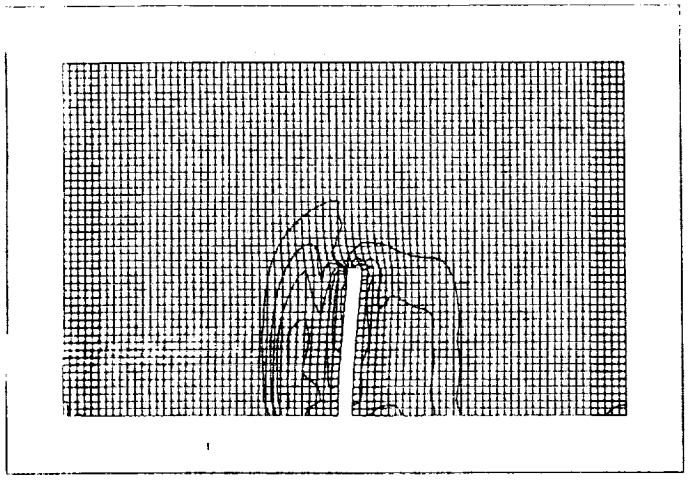
Figure 5.10: Sigma - Y stress contours for the final deformed configuration of the flexible wall using the local remeshing method.



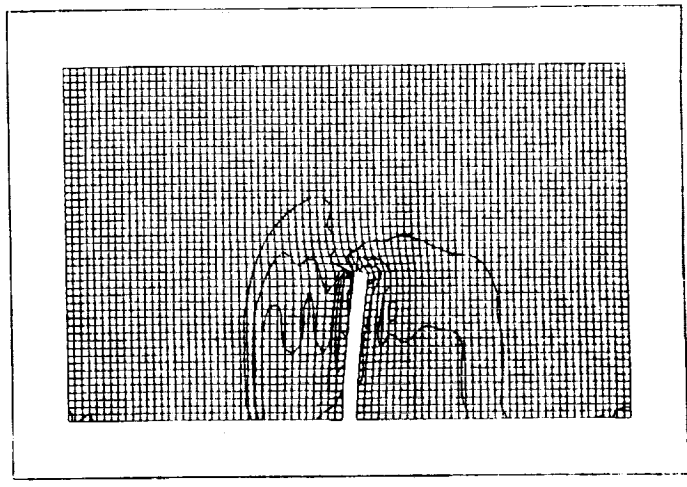
0



20

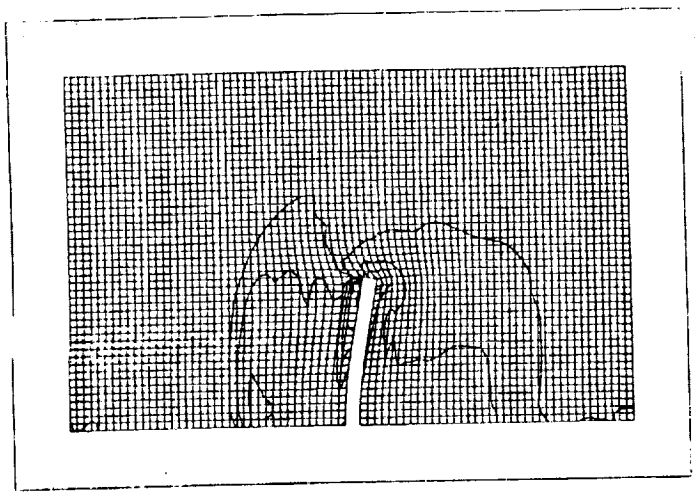


40

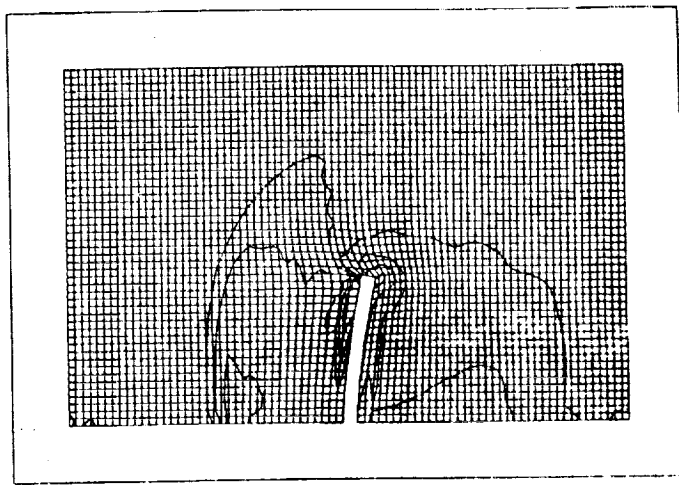


60

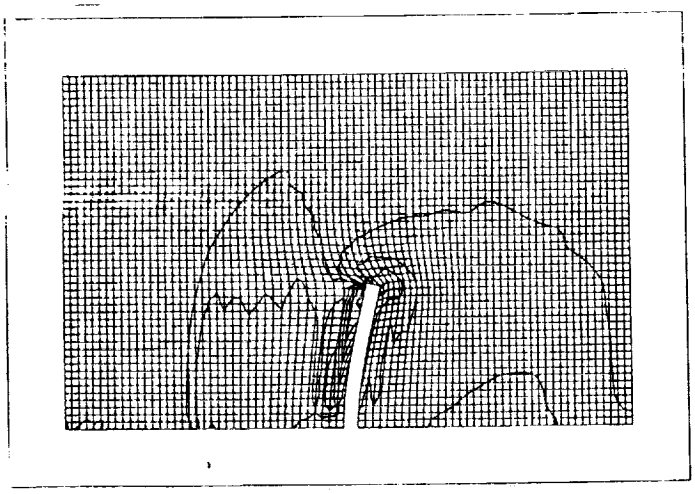
Figure 5.11a: Evolution of density contours for the lox-post simulation using the local remeshing method. Numbers under figures refer to time step.



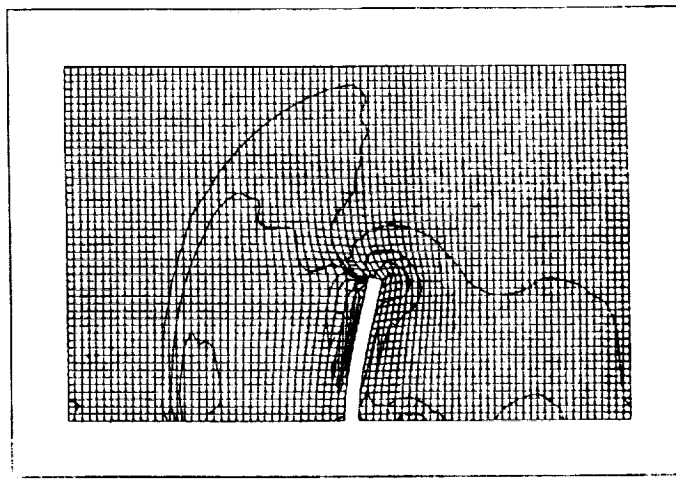
80



100

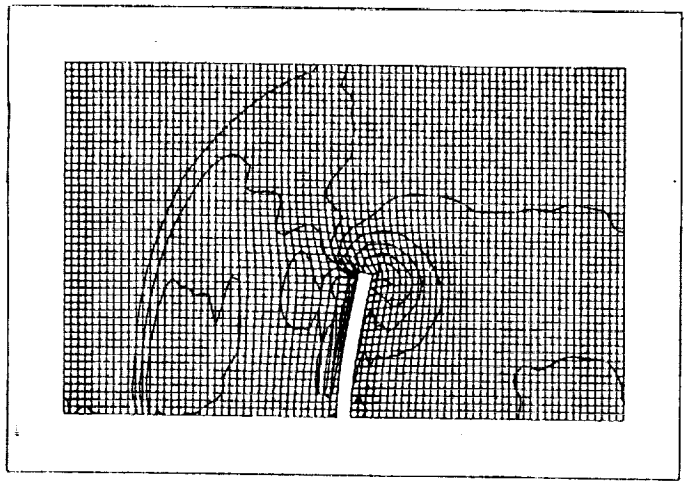


120

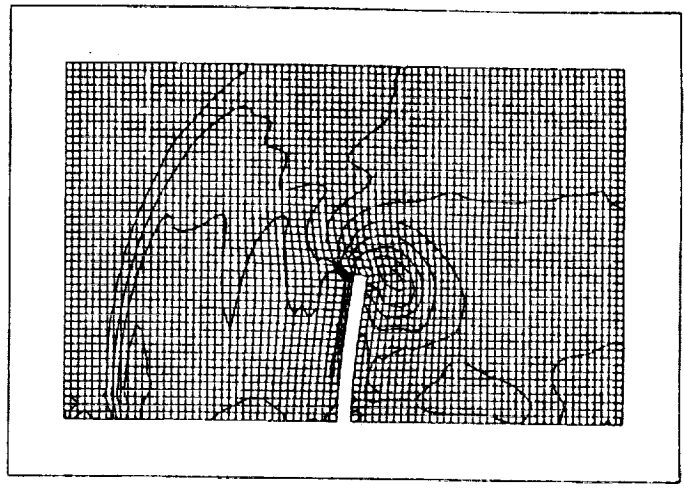


140

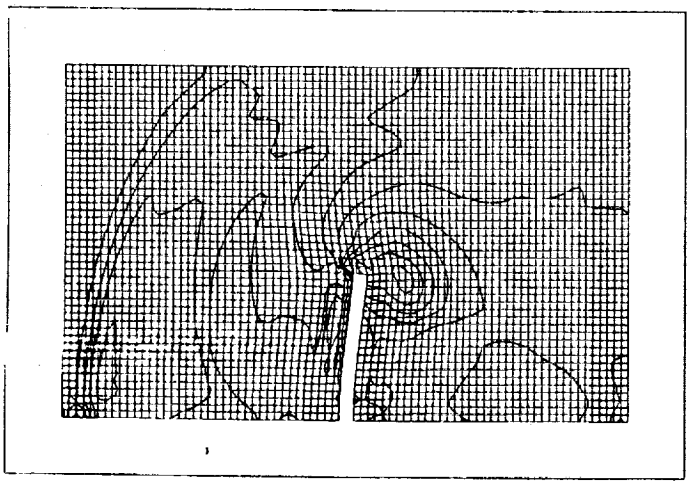
Figure 5.11b: Evolution of density contours for the lox-post simulation using the local remeshing method. Numbers under figures refer to time step.



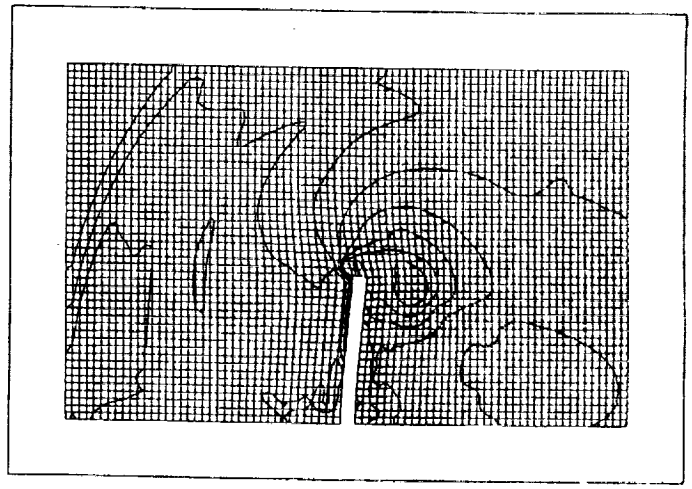
160



180

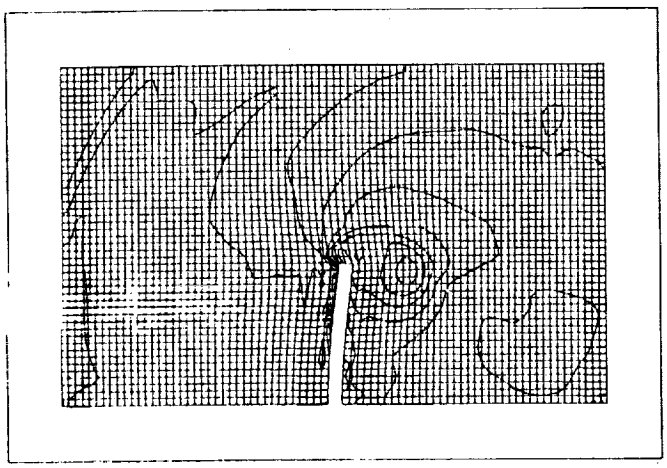


200

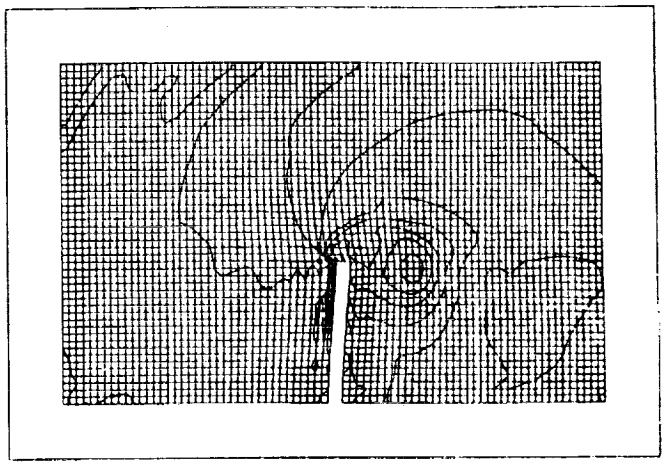


220

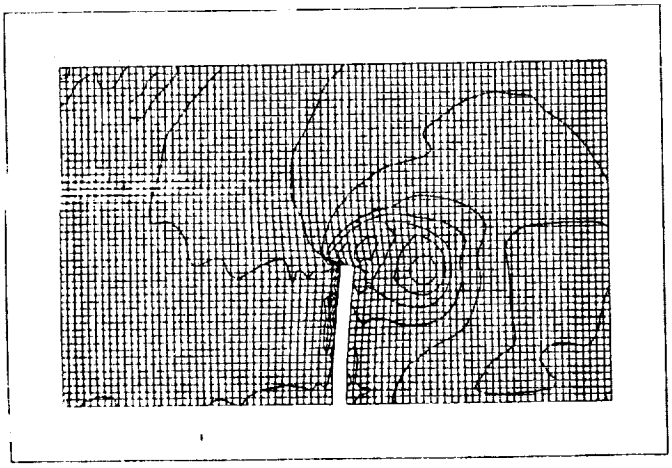
Figure 5.11c: Evolution of density contours for the lox-post simulation using the local remeshing method. Numbers under figures refer to time step.



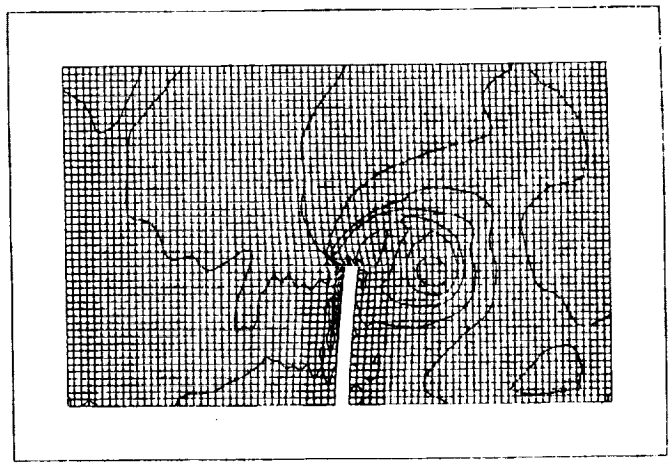
240



260

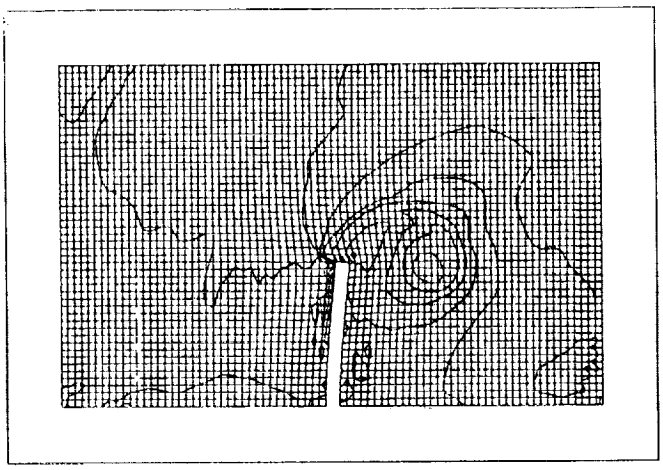


280

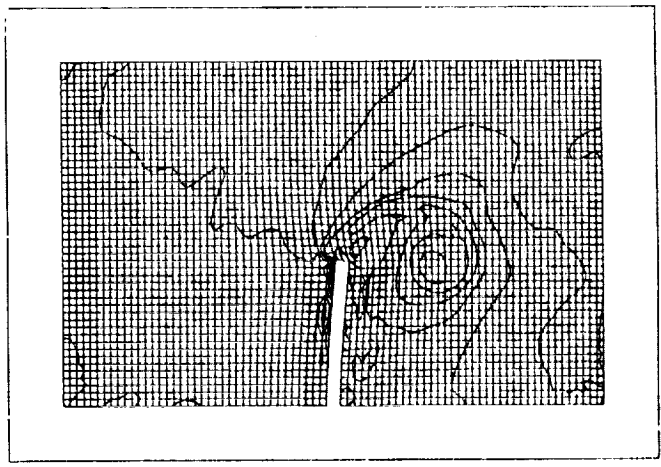


300

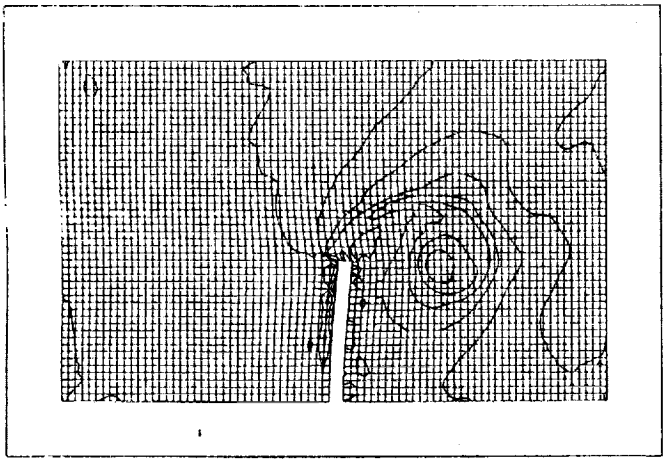
Figure 5.11d: Evolution of density contours for the lox-post simulation using the local remeshing method. Numbers under figures refer to time step.



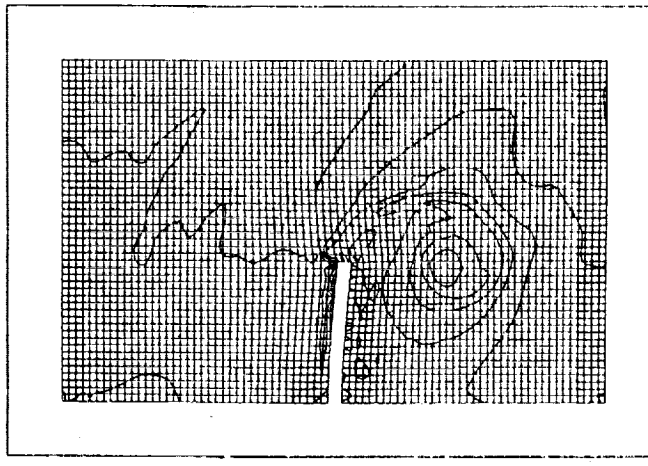
320



340

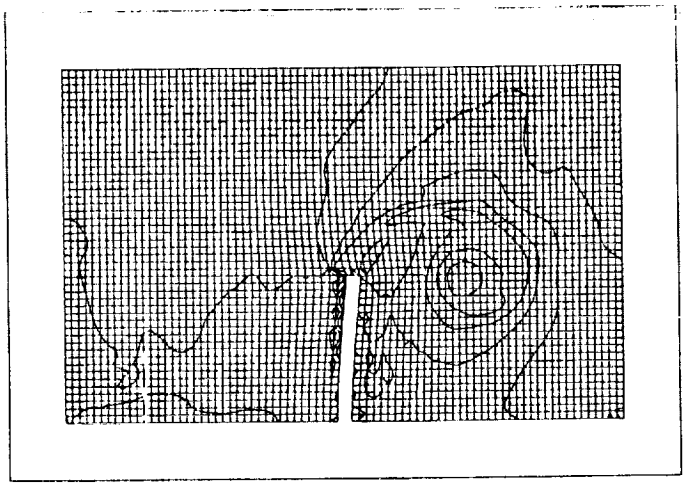


360

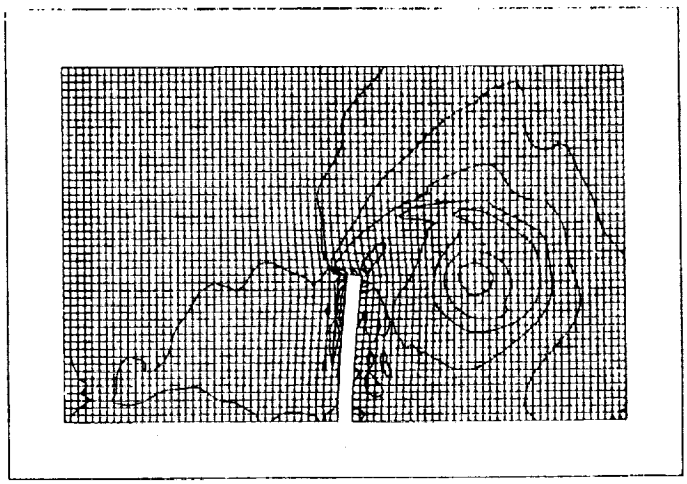


380

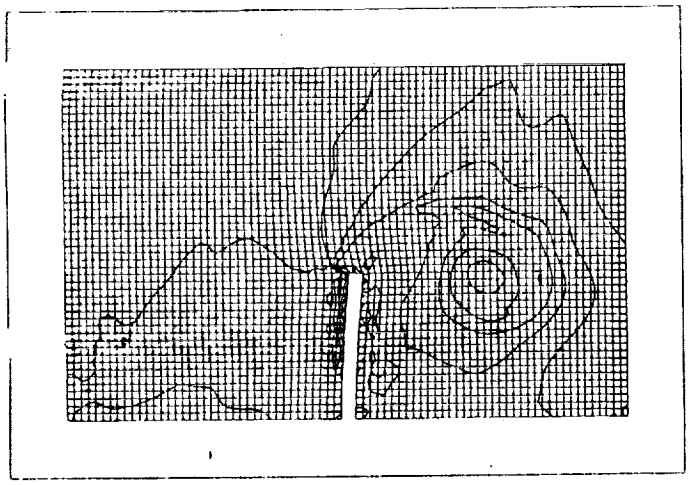
Figure 5.11e: Evolution of density contours for the lox-post simulation using the local remeshing method. Numbers under figures refer to time step.



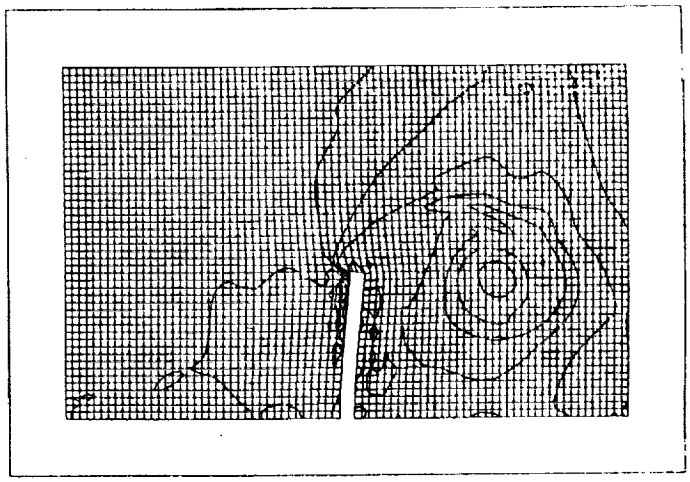
400



420

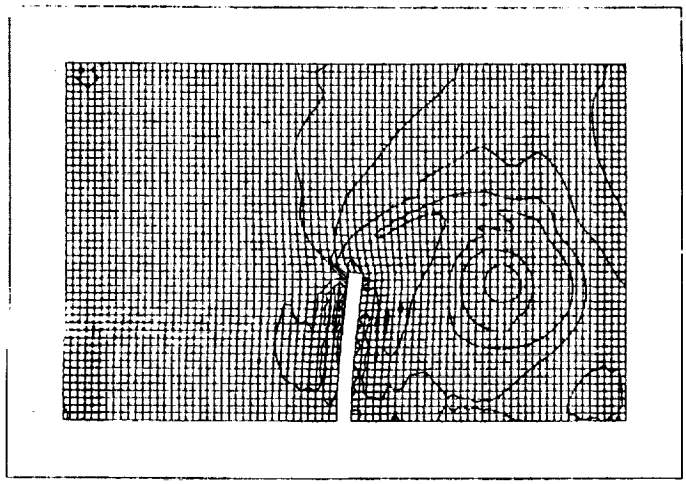


440

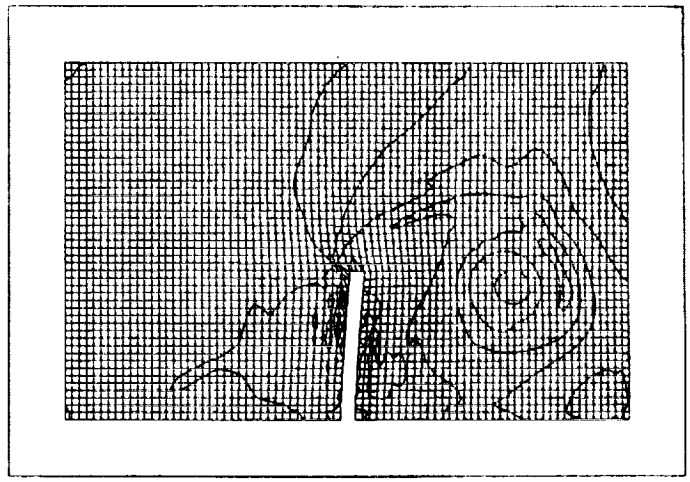


460

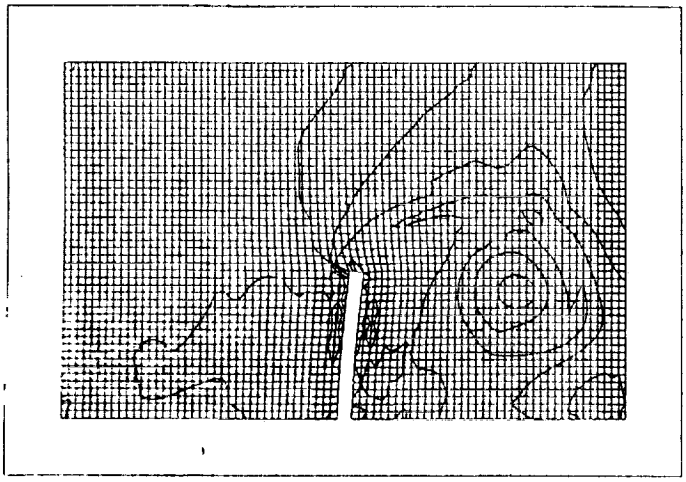
Figure 5.11f: Evolution of density contours for the lox-post simulation using the local remeshing method. Numbers under figures refer to time step.



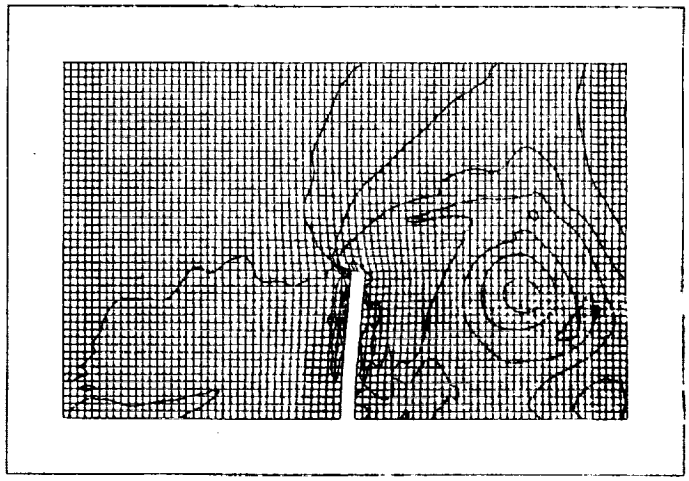
480



500

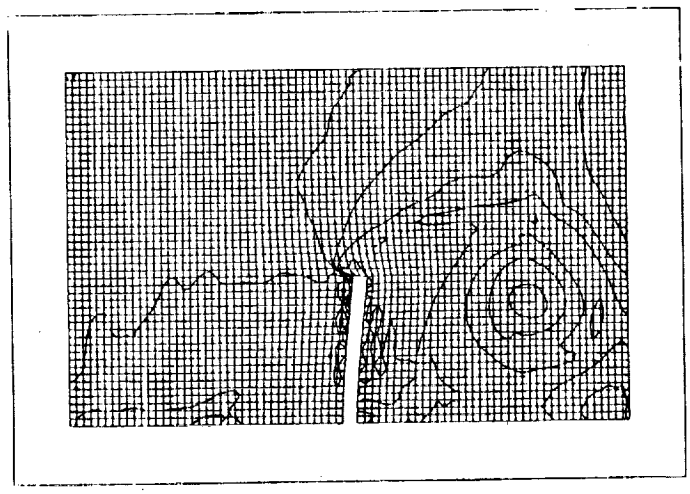


520

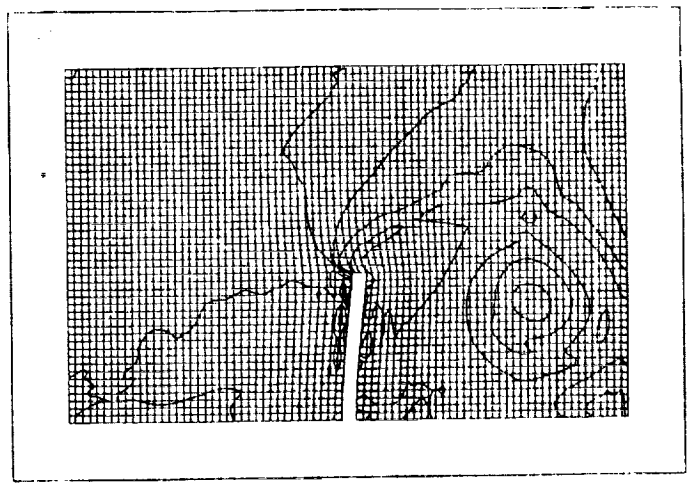


540

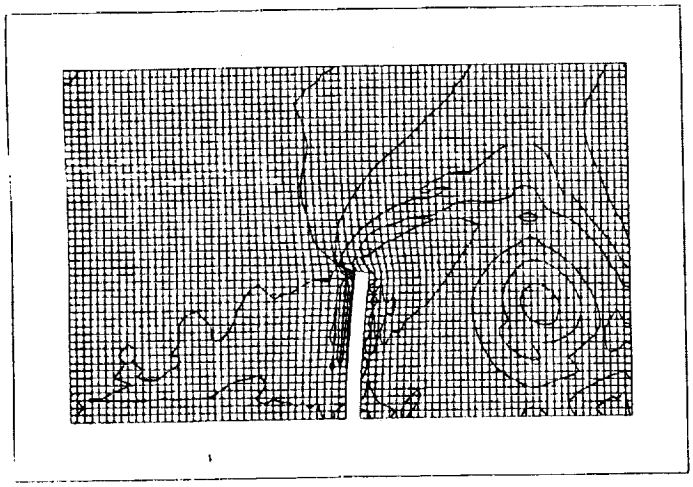
Figure 5.11g: Evolution of density contours for the lox-post simulation using the local remeshing method. Numbers under figures refer to time step.



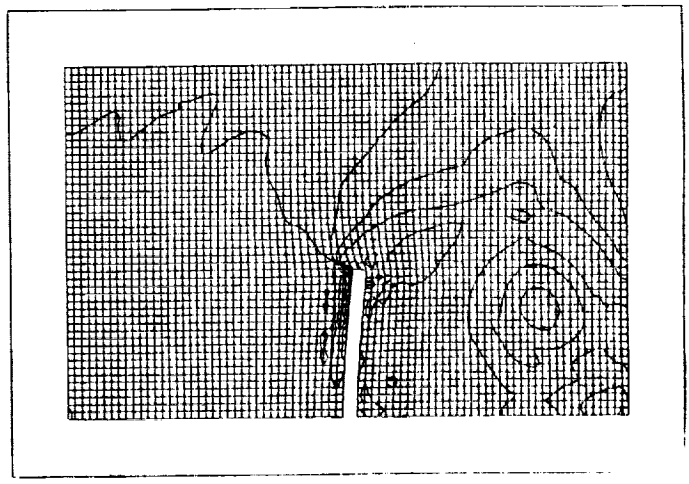
560



580

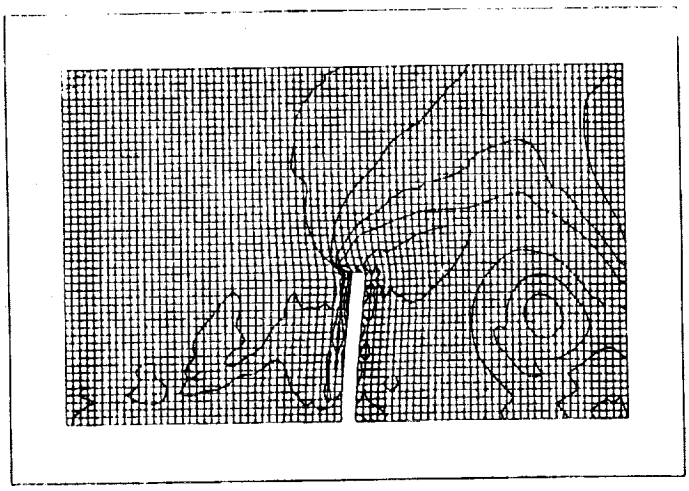


600

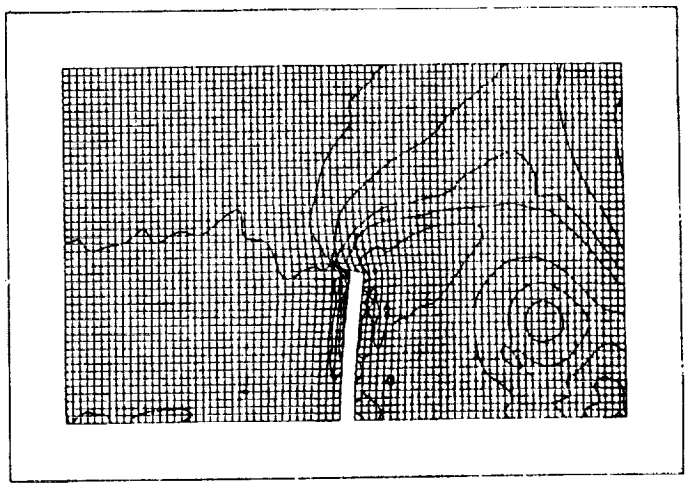


620

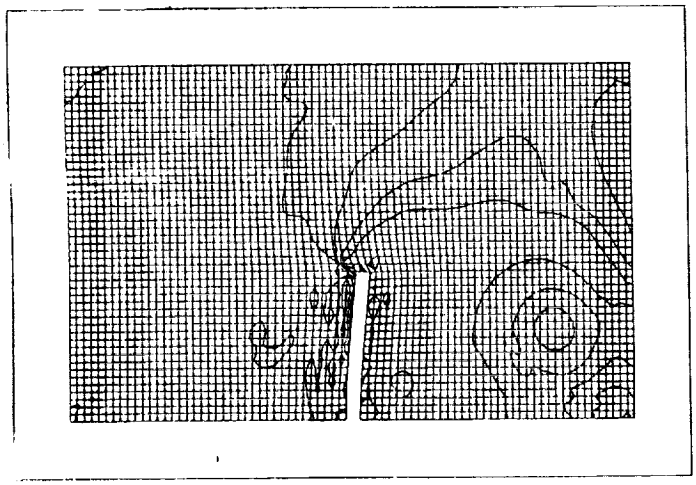
Figure 5.11h: Evolution of density contours for the lox-post simulation using the local remeshing method. Numbers under figures refer to time step.



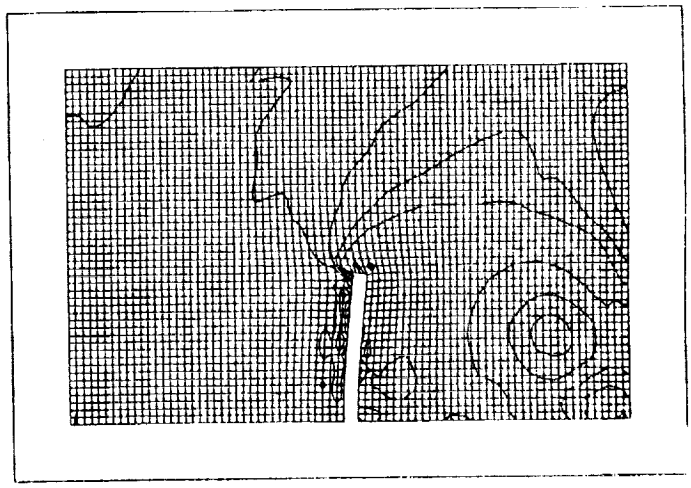
640



660

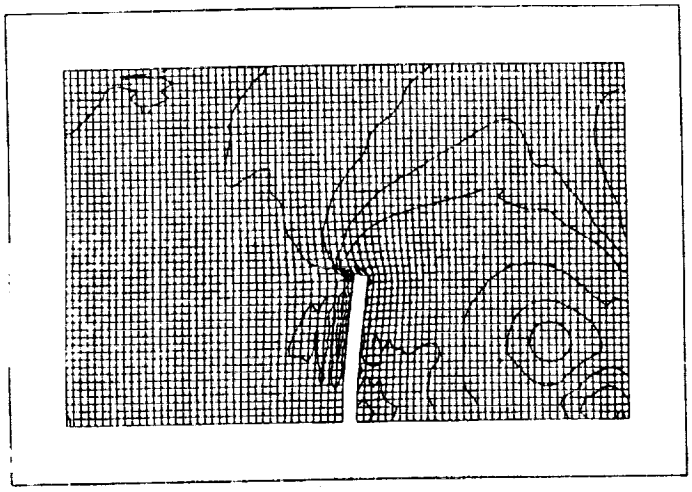


680

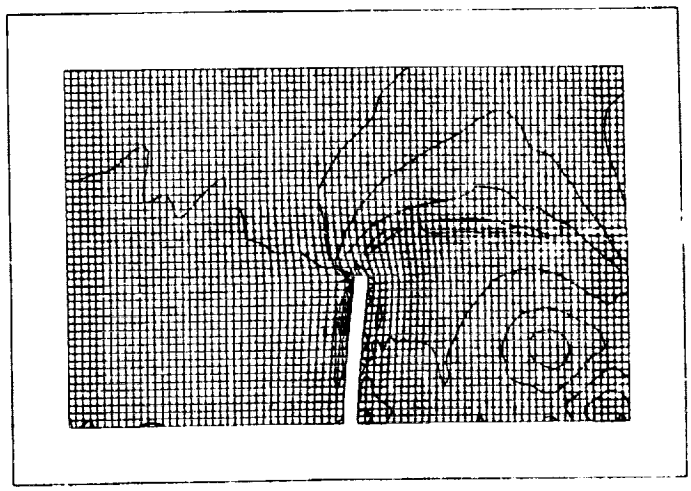


700

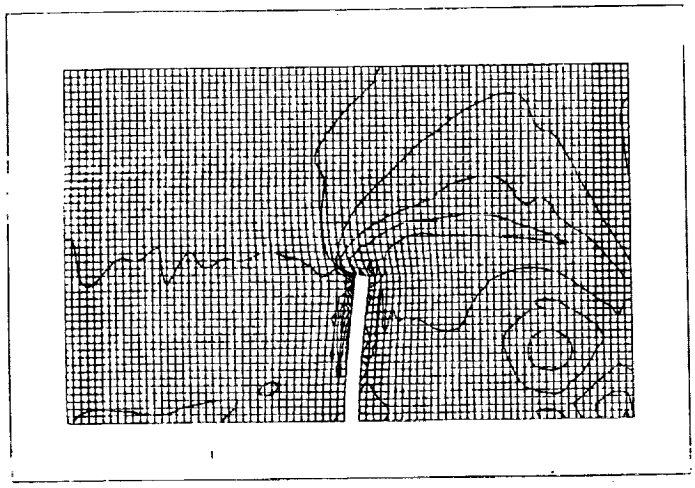
Figure 5.11i: Evolution of density contours for the lox-post simulation using the local remeshing method. Numbers under figures refer to time step.



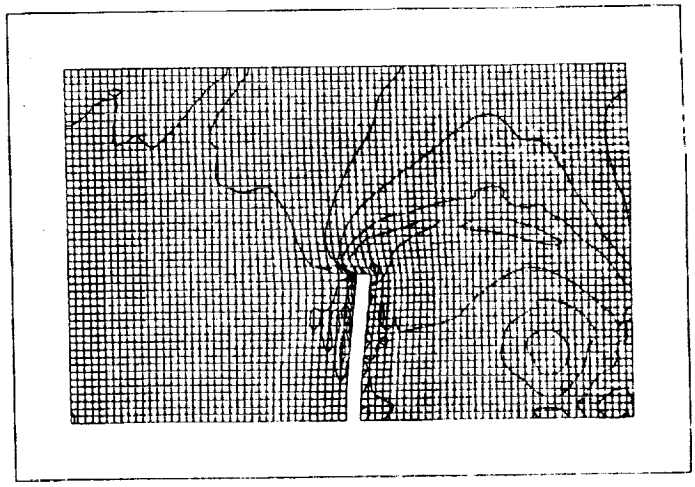
720



740

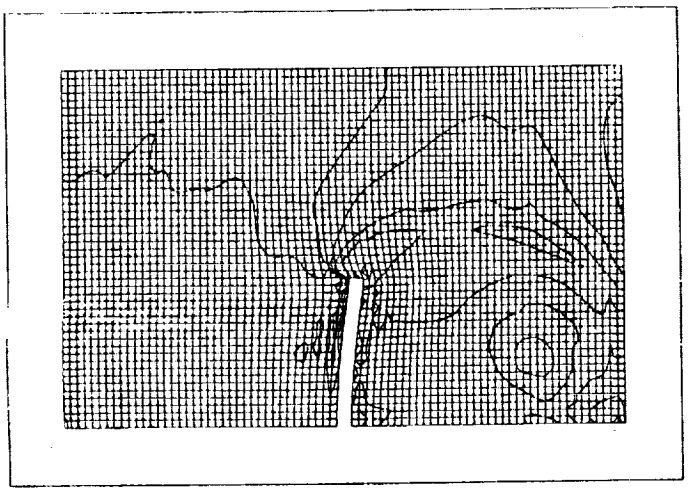


760

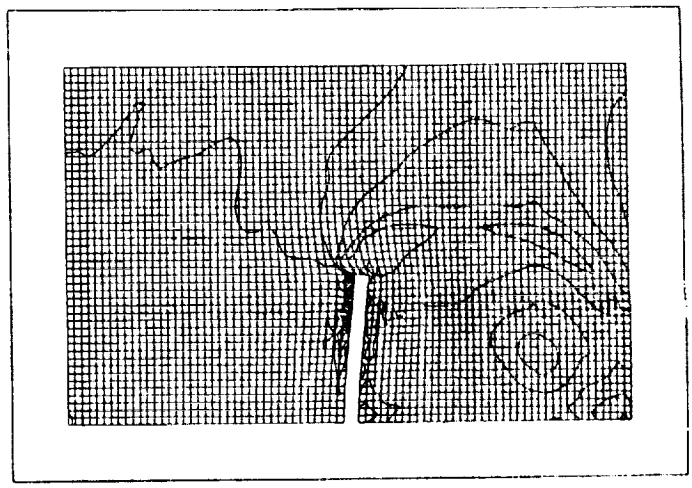


780

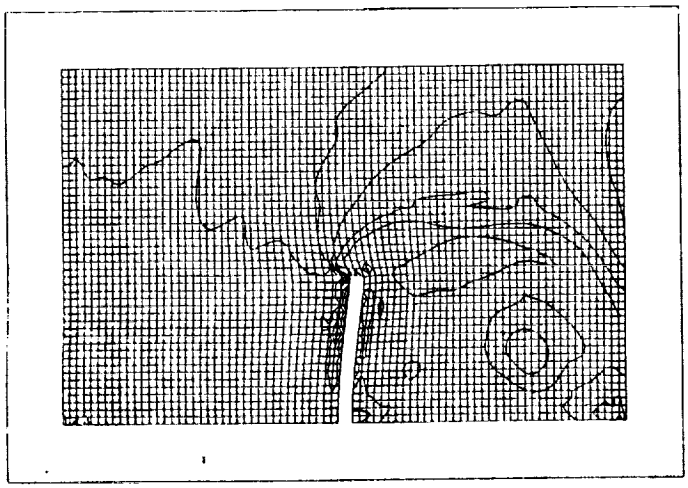
Figure 5.11j: Evolution of density contours for the lox-post simulation using the local remeshing method. Numbers under figures refer to time step.



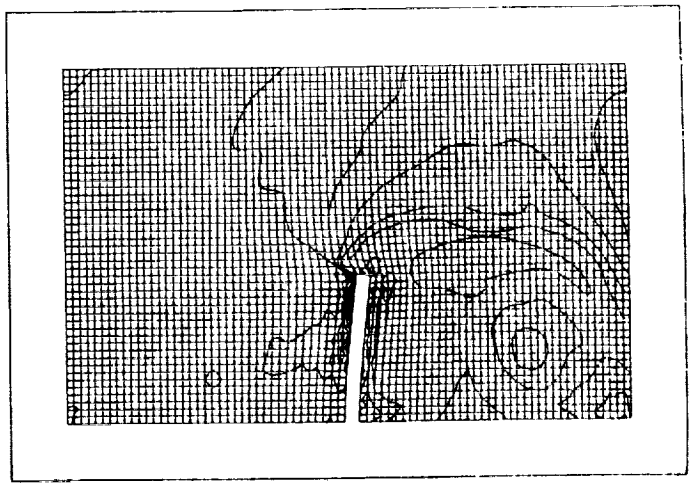
800



820

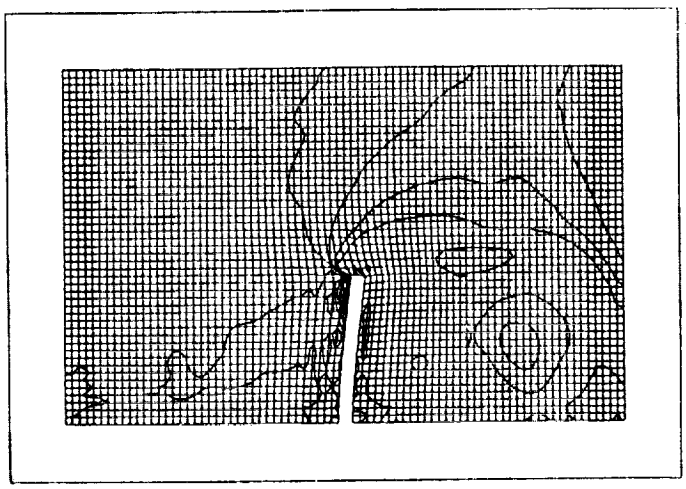


840

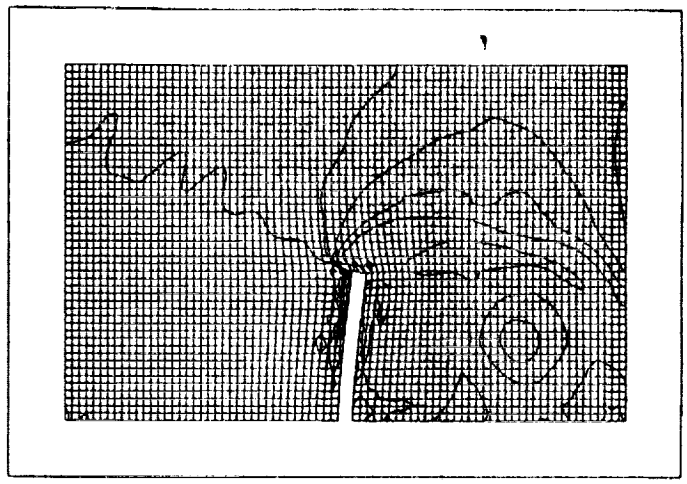


860

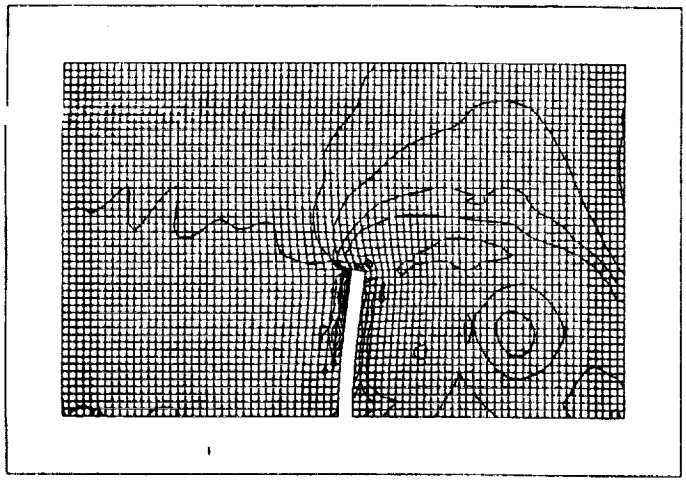
Figure 5.11k: Evolution of density contours for the lox-post simulation using the local remeshing method. Numbers under figures refer to time step.



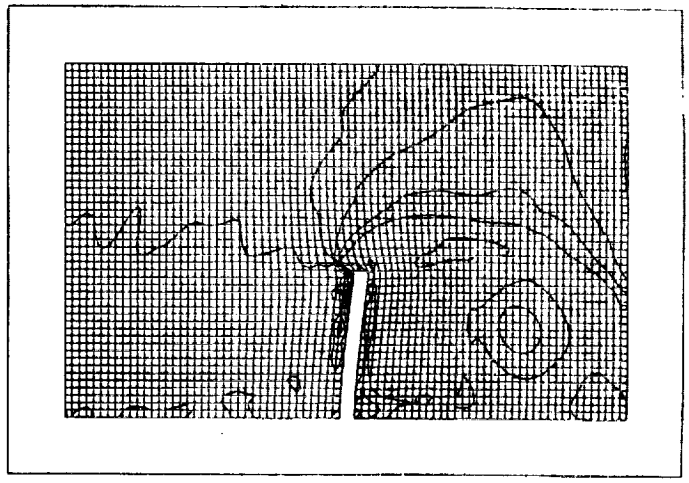
880



900

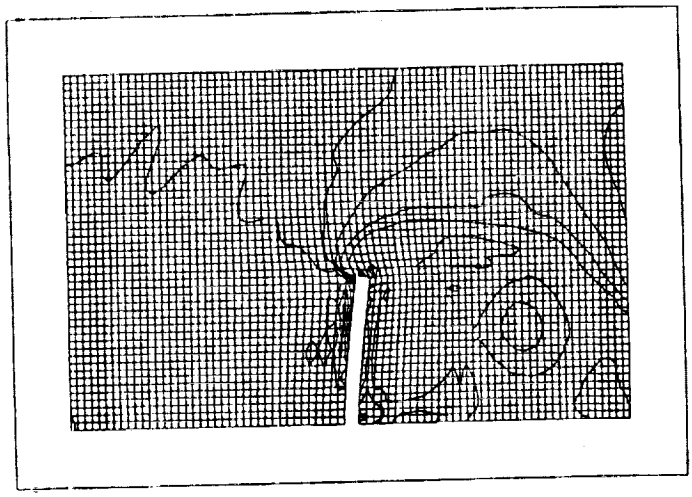


920

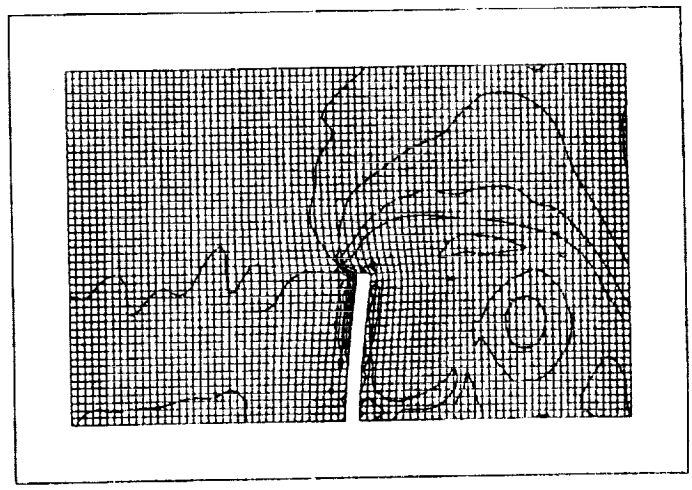


940

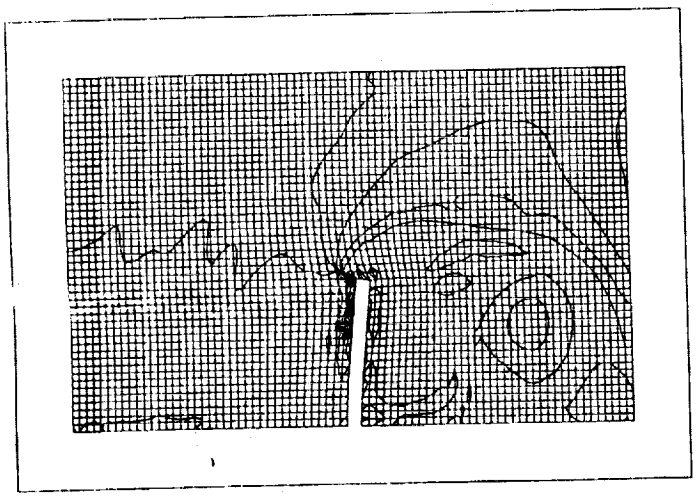
Figure 5.11l: Evolution of density contours for the lox-post simulation using the local remeshing method. Numbers under figures refer to time step.



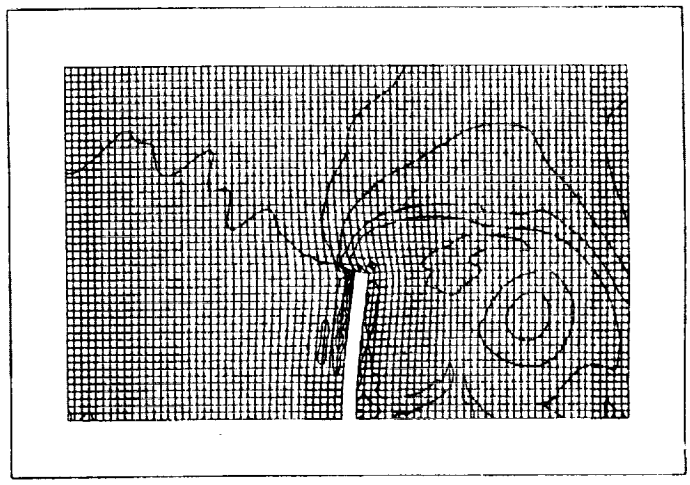
960



980



1000



1020

Figure 5.11m: Evolution of density contours for the lox-post simulation using the local remeshing method. Numbers under figures refer to time step.

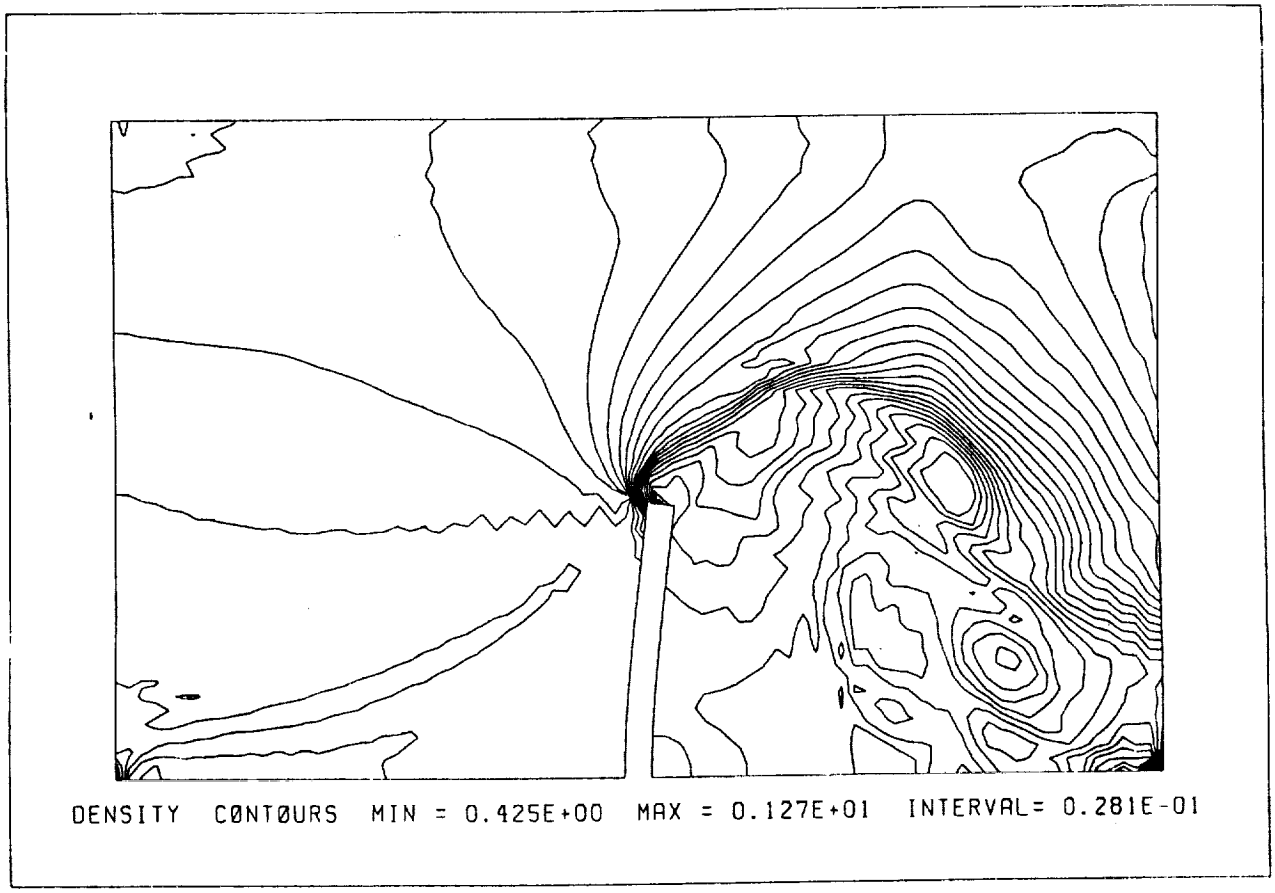


Figure 5.12: Density contours for the final configuration of the lox-post simulation.

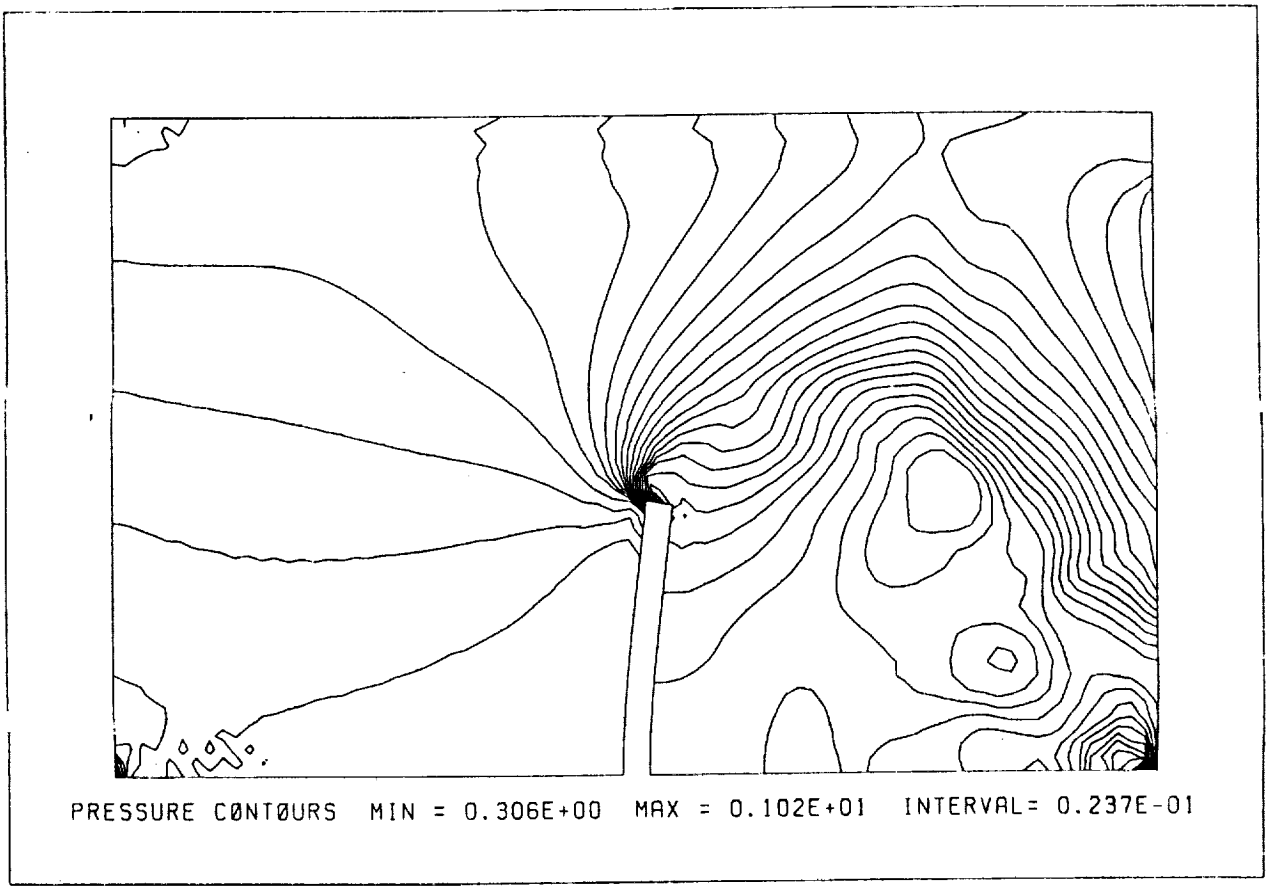


Figure 5.13: Pressure contours for the final configuration of the lox-post simulation.

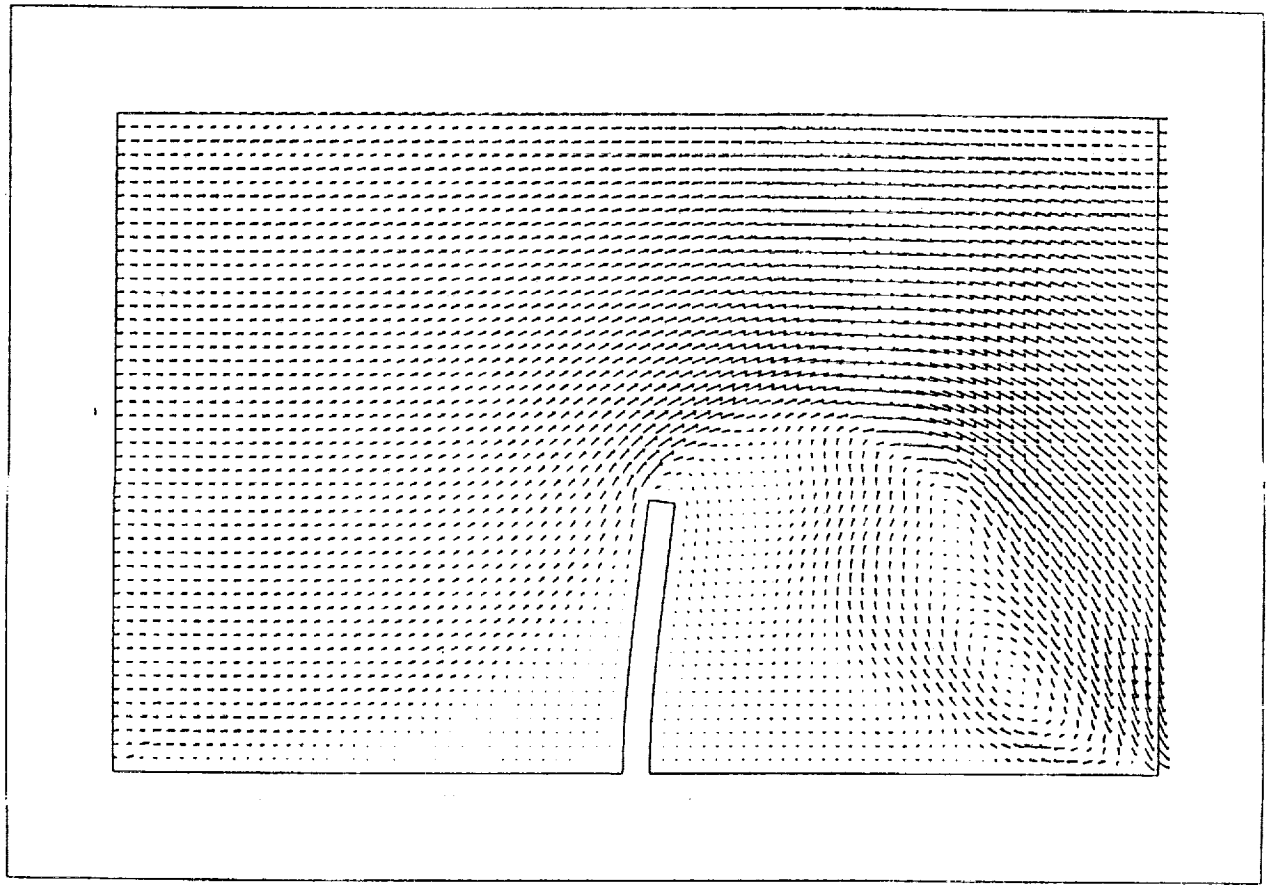
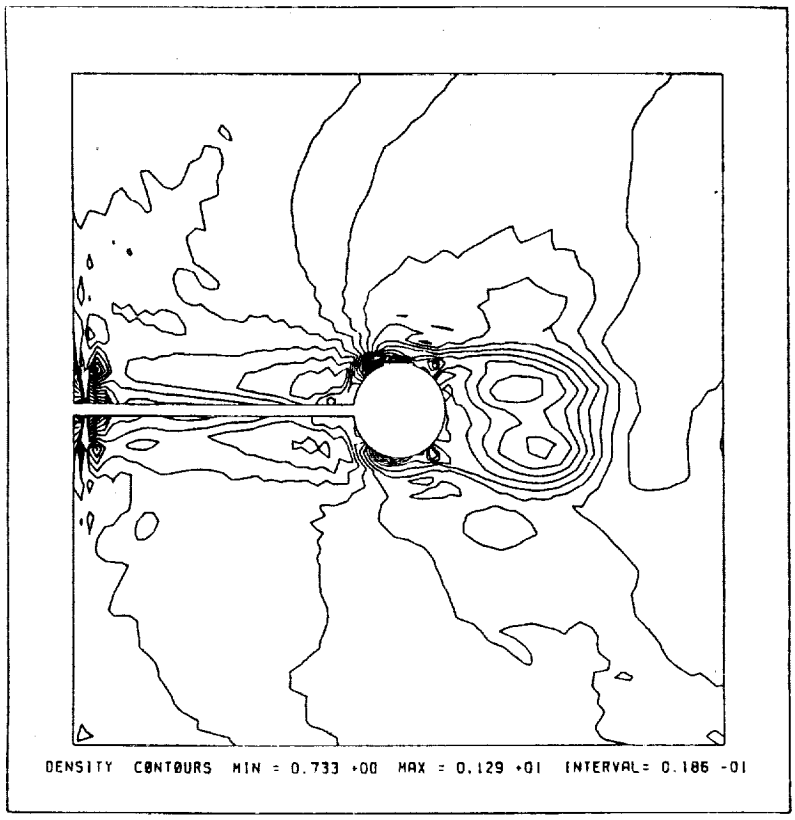
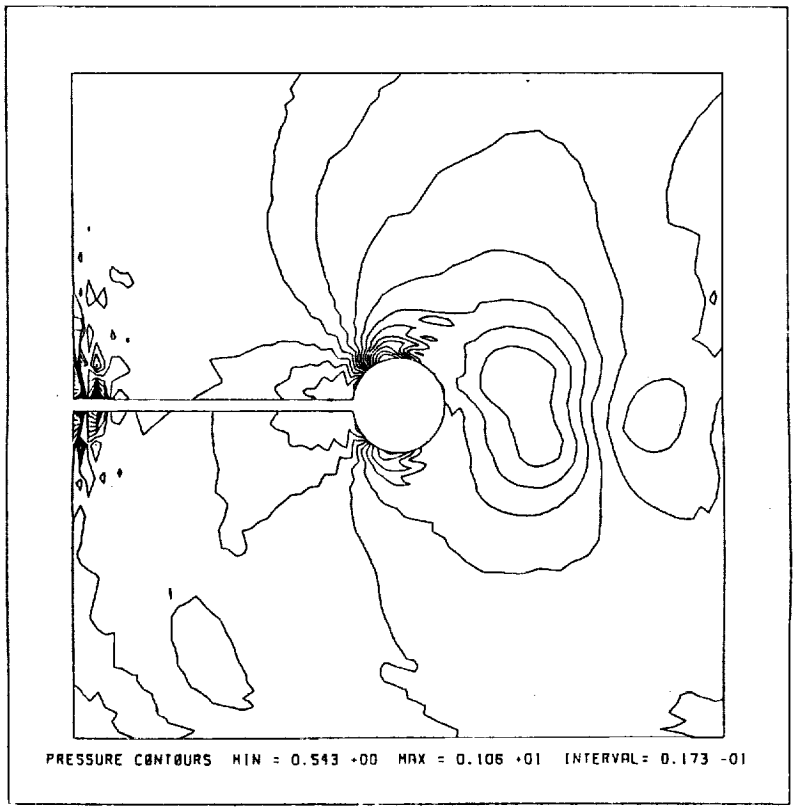


Figure 5.14: Velocity vectors for the final configuration of the lox-post simulation.

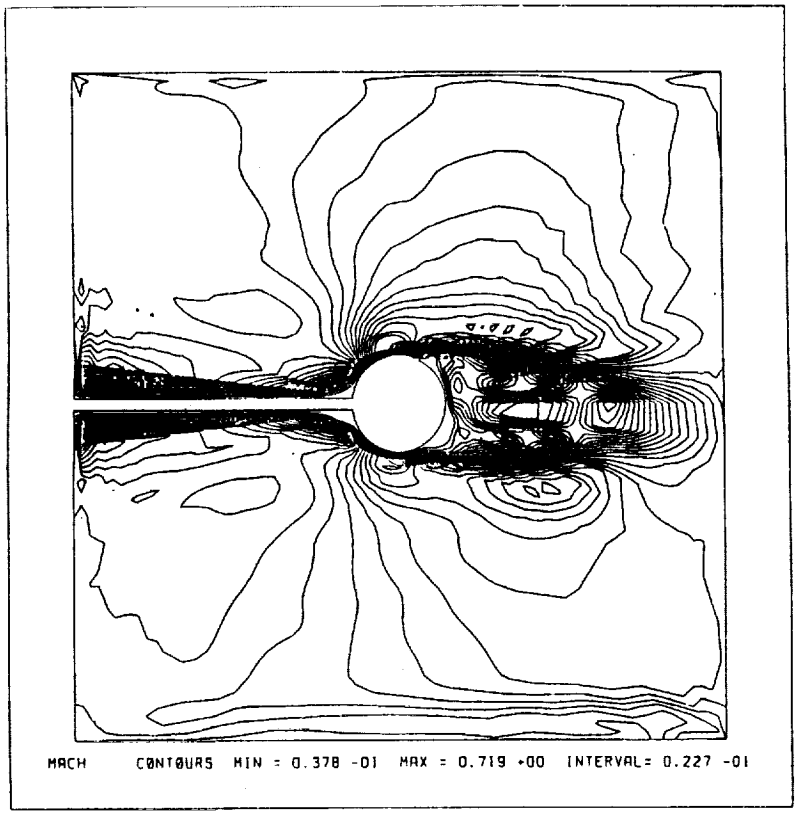


a

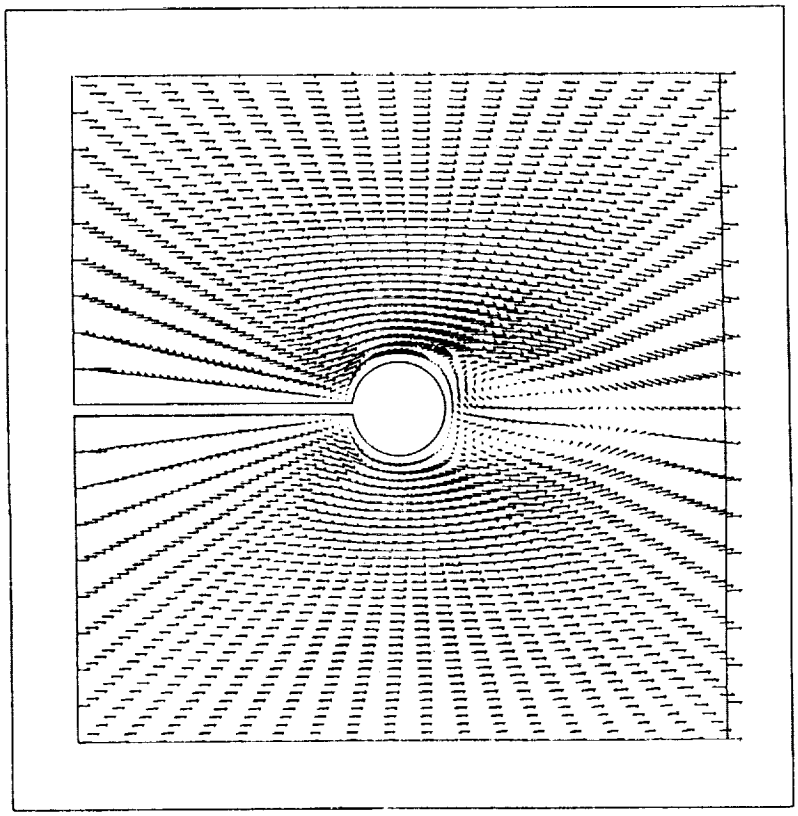


b

Figure 5.15a,b: Results for the rigid cylinder, flexible shaft problem using the quasi-steady state method—nondeformed configuration. (a) Density contours, (b) pressure contours.



a



b

Figure 5.15c,d: Results for the rigid cylinder, flexible shaft problem using the quasi-steady state method—nondeformed configuration. (a) Mach contours, (d) velocity vectors.

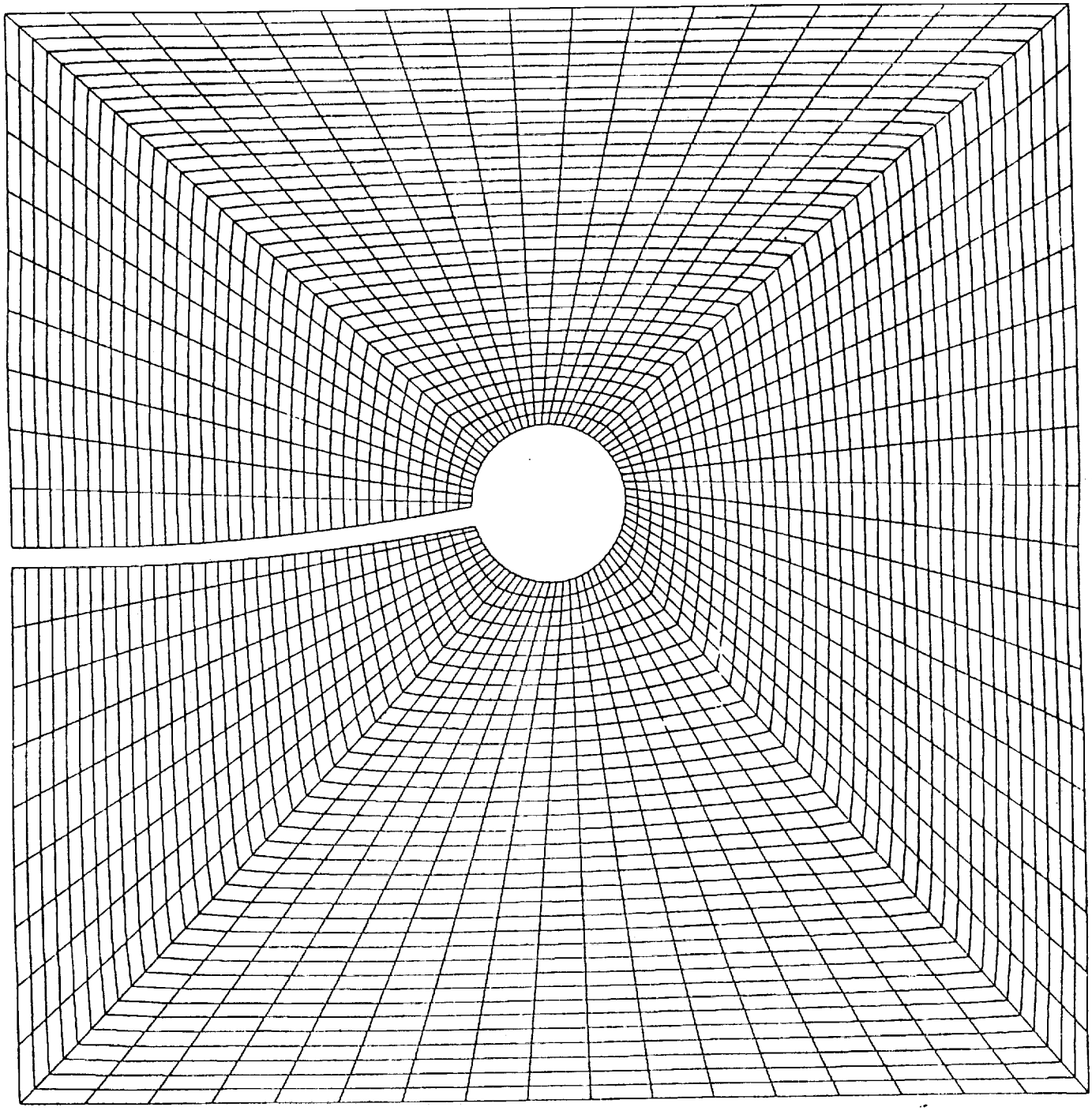
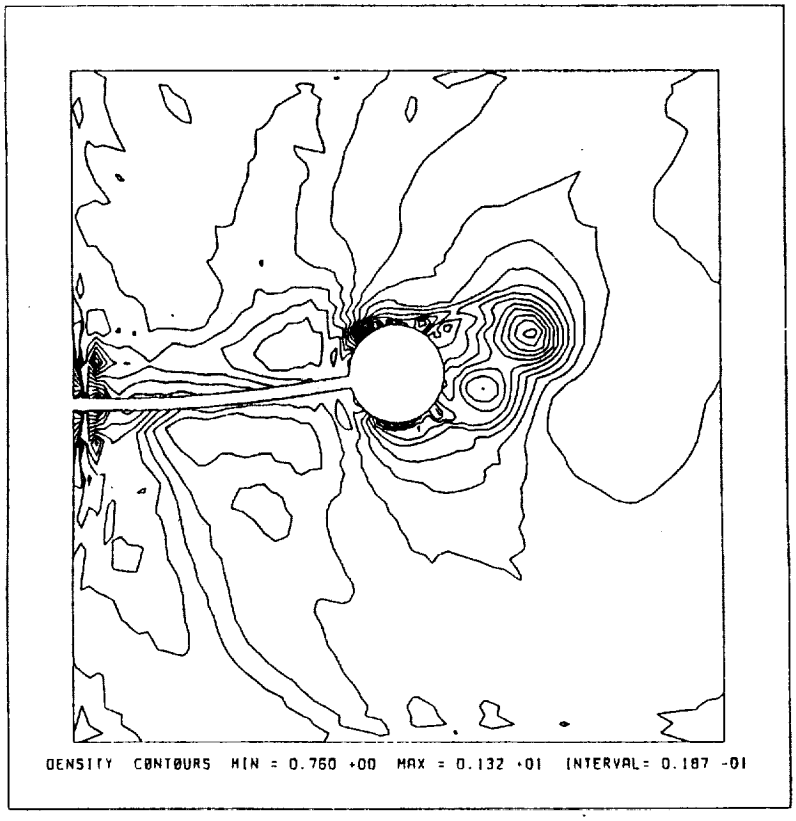
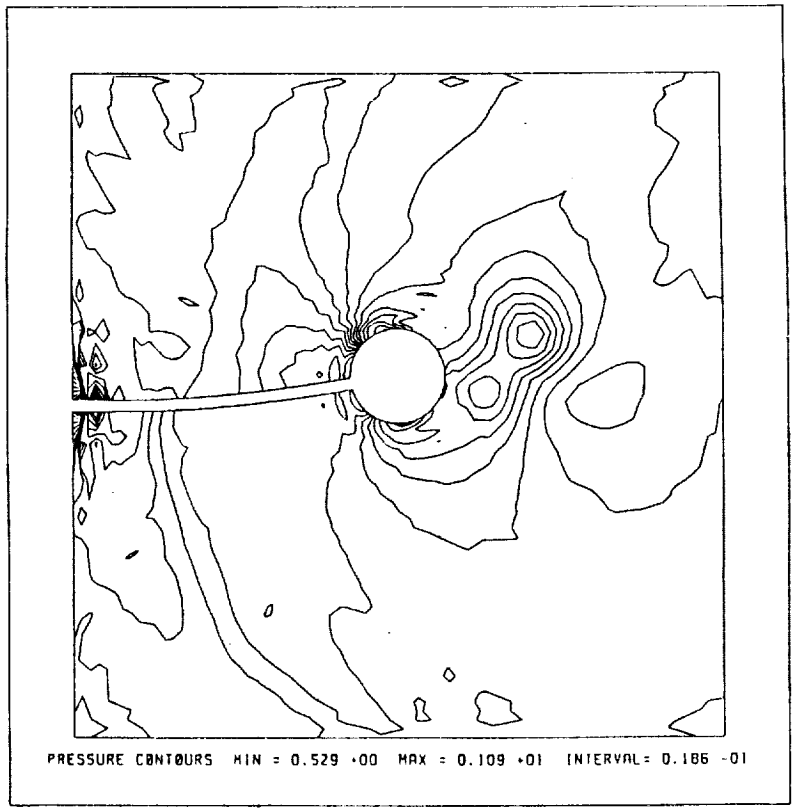


Figure 5.16a: First deformed configuration of the computational domain for the rigid cylinder, flexible shaft problem using the quasi-steady state method.

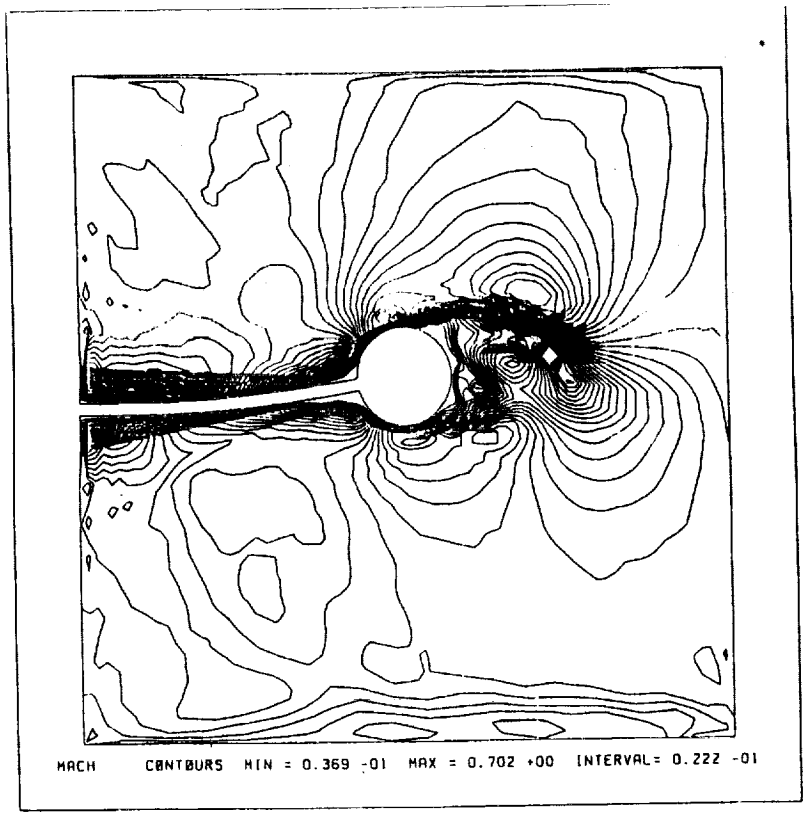


b

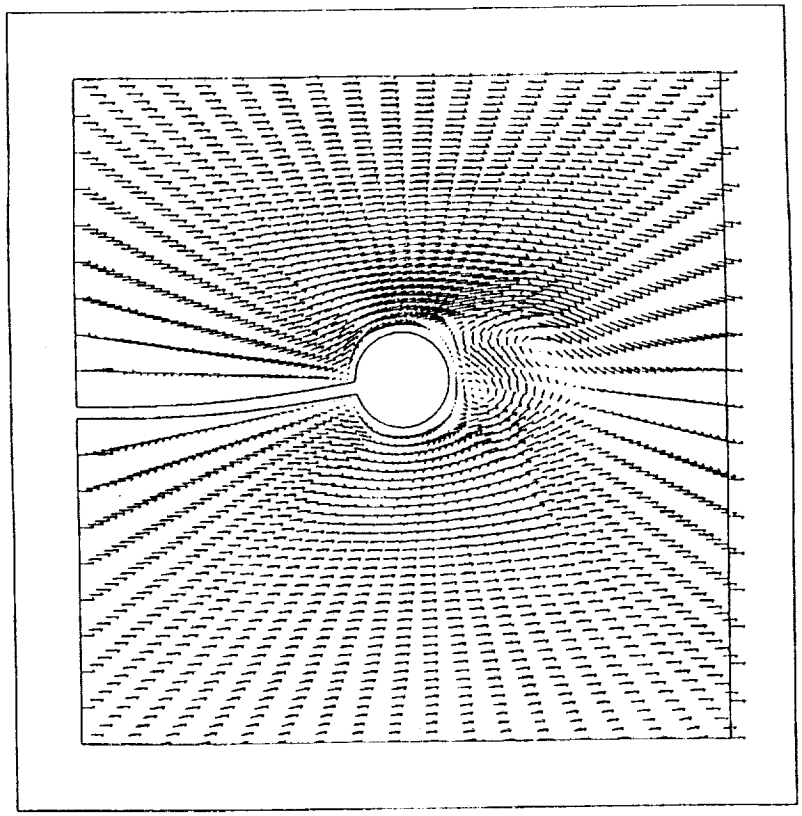


c

Figure 5.16b,c: Results for the rigid cylinder, flexible shaft problem using the quasi–steady method—first deformed configuration. (b) Density contours, (c) pressure contours.



d



e

Figure 5.16d,e: Results for the rigid cylinder, flexible shaft problem using the quasi-steady state method—first deformed configuration. (d) Mach contours, (e) velocity vectors.

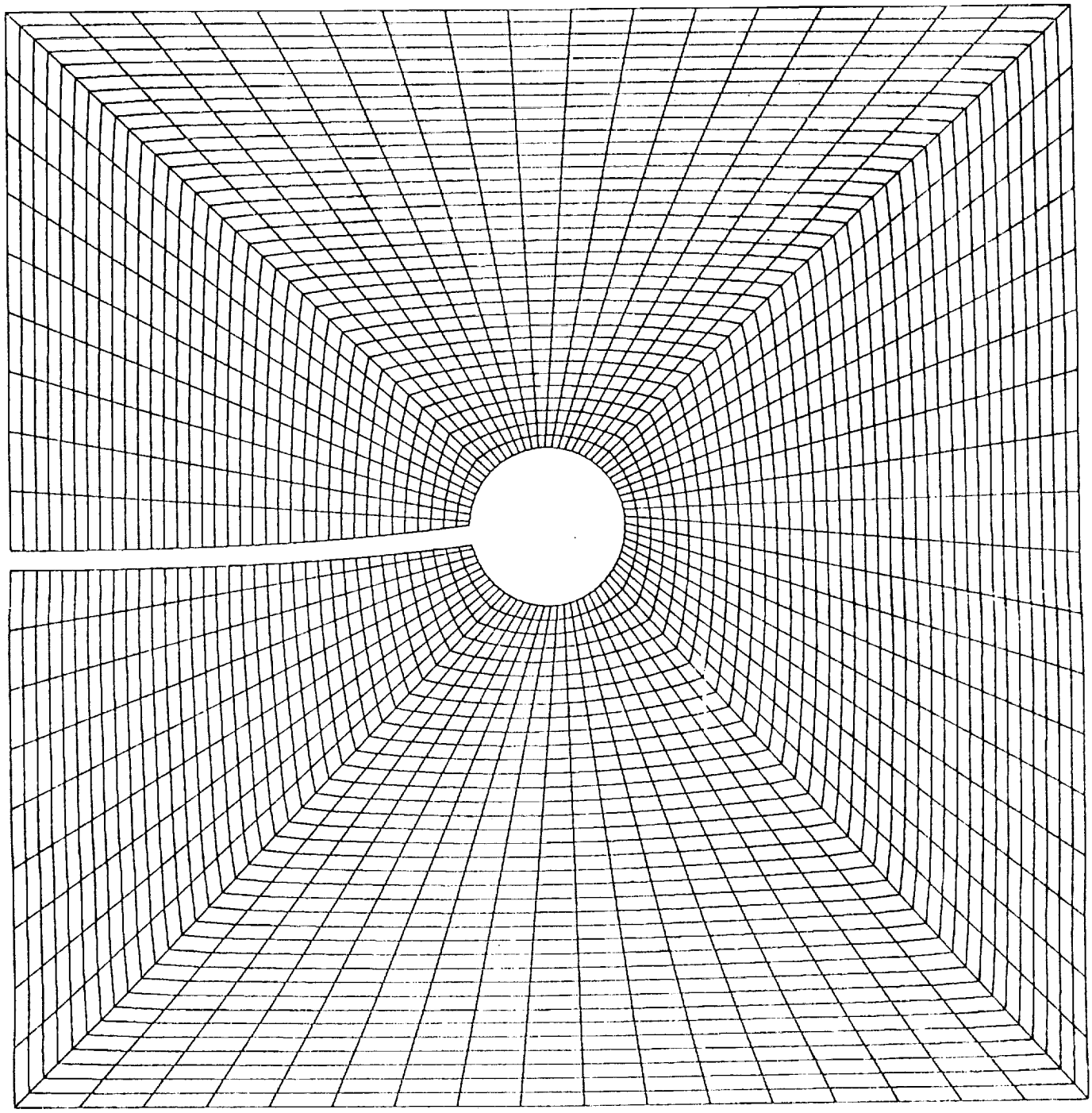
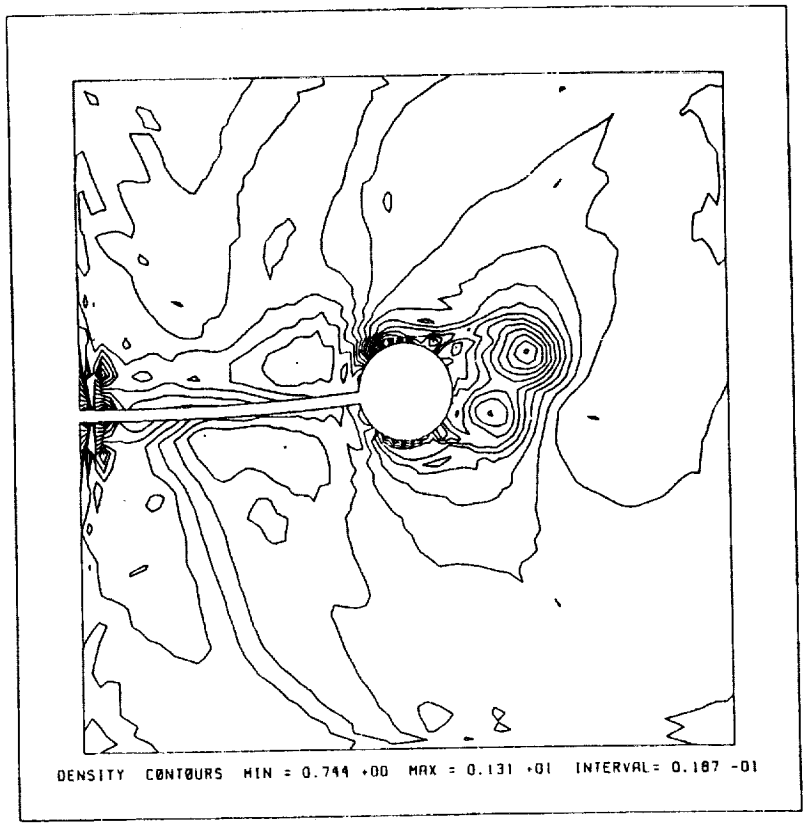
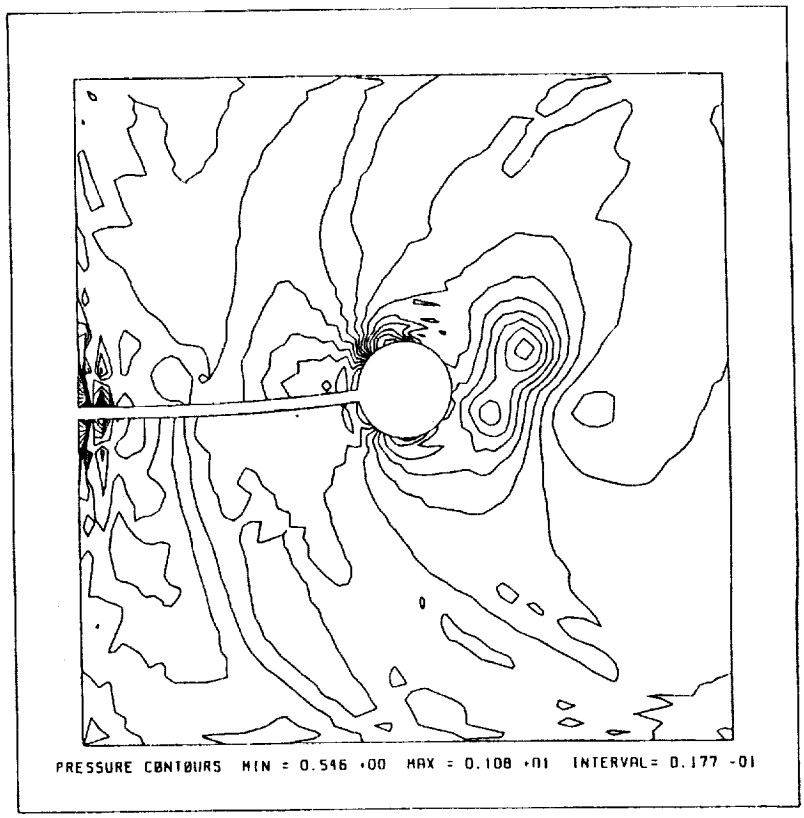


Figure 5.17a: Second deformed configuration of the computational domain for the rigid cylinder, flexible shaft problem using the quasi-steady state method.

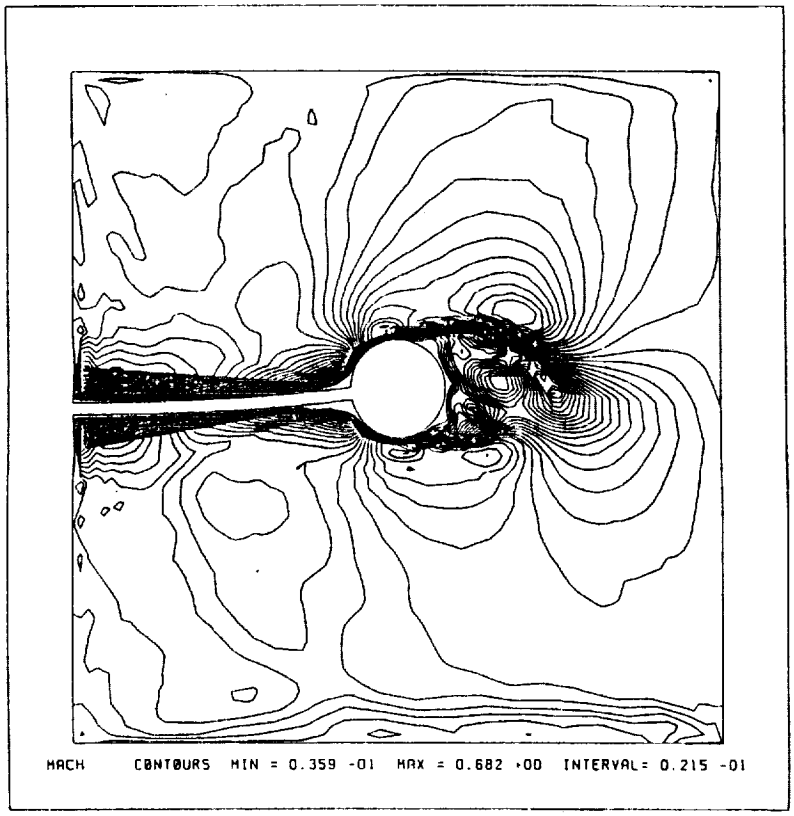


b

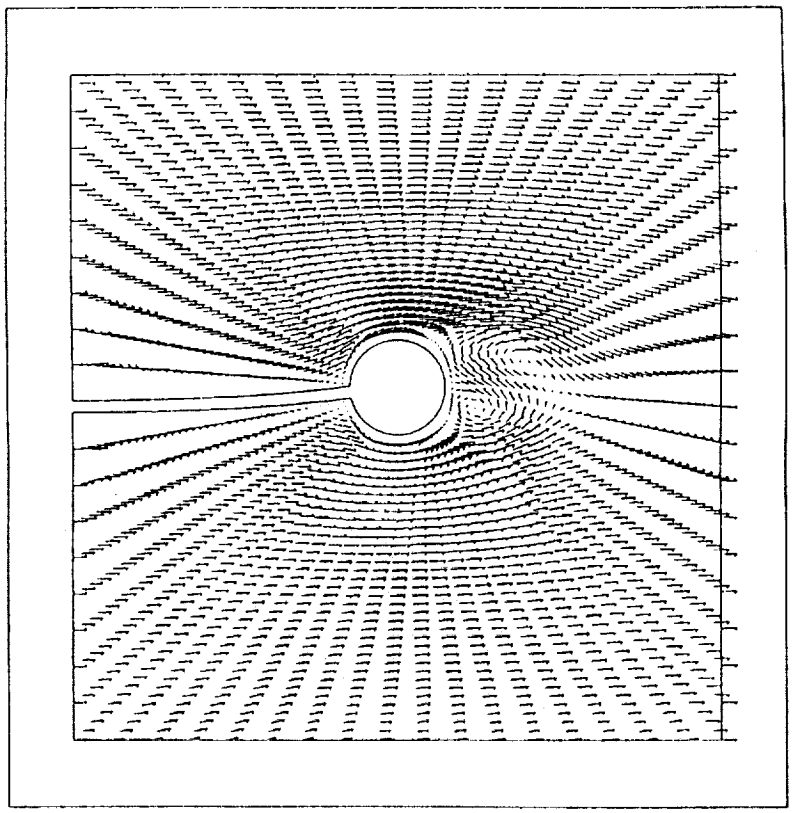


c

Figure 5.17b,c: Results for the rigid cylinder, flexible shaft problem using the quasi-steady state method—second deformed configuration. (b) Density contours, (c) pressure contours.



d



e

Figure 5.17d,e: Results for the rigid cylinder, flexible shaft problem using the quasi-steady state method—second deformed configuration. (d) Mach contours, (e) velocity vectors.

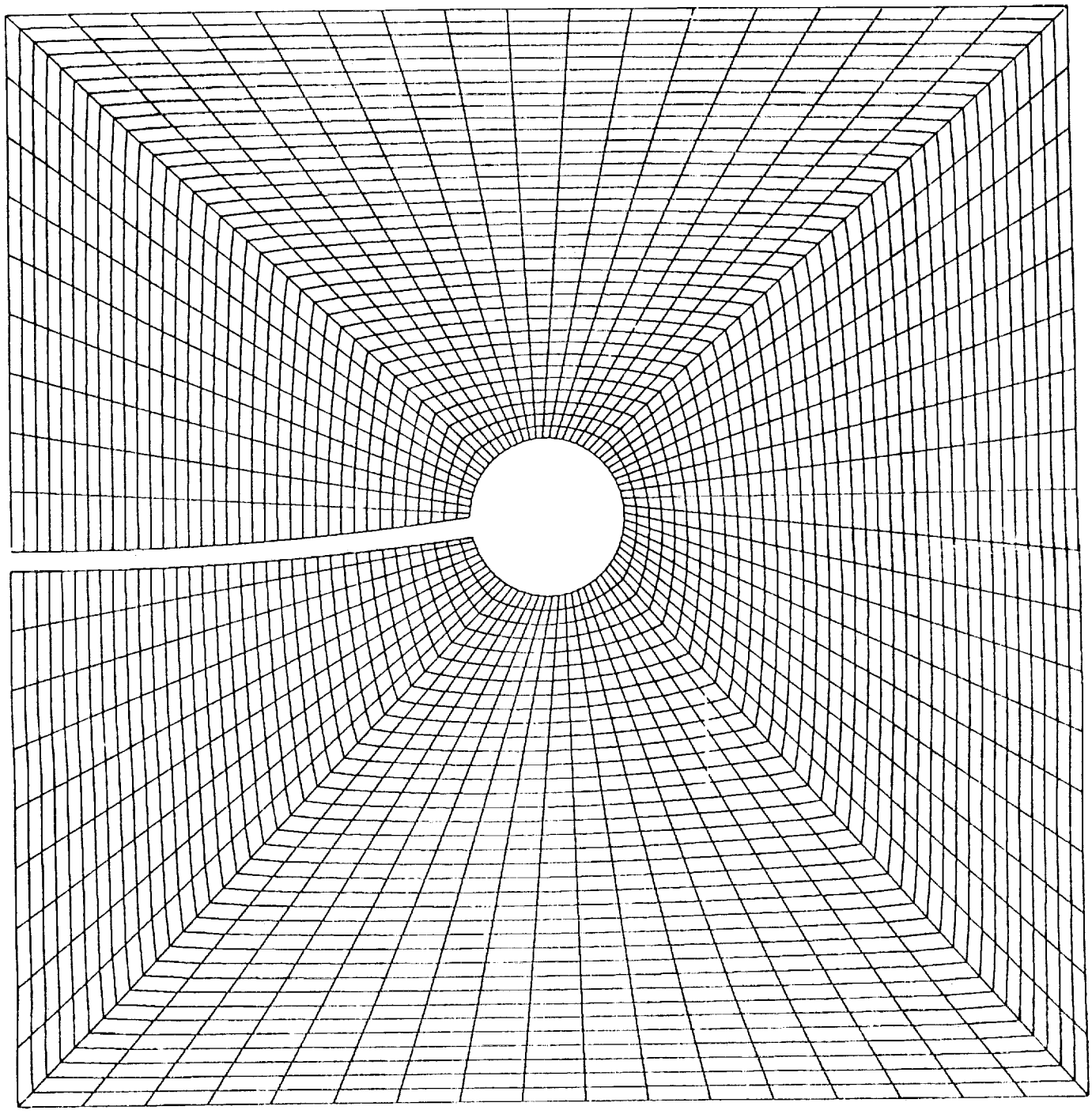
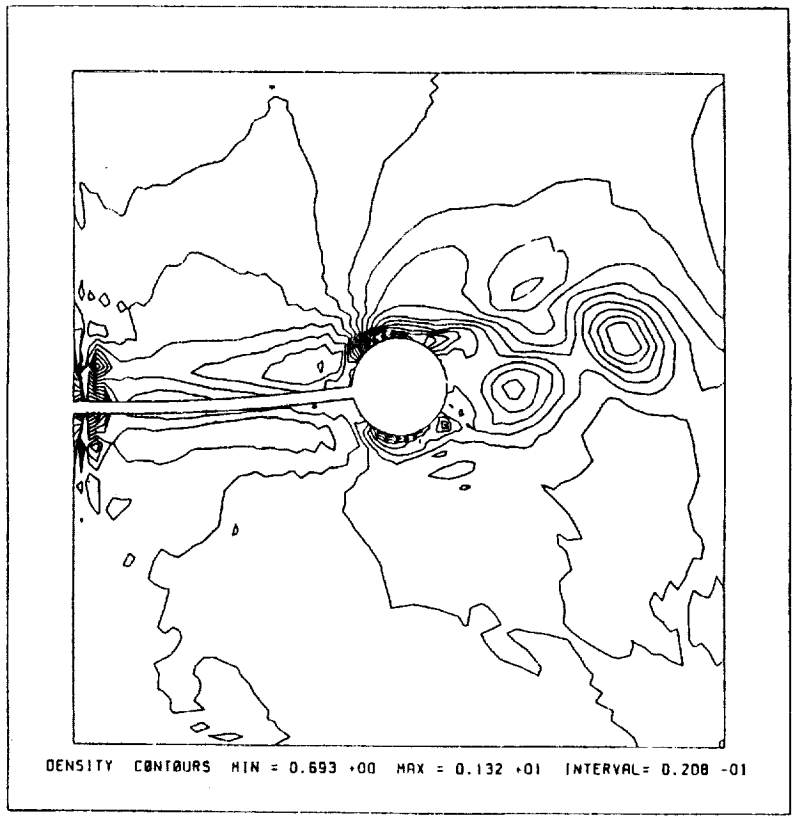
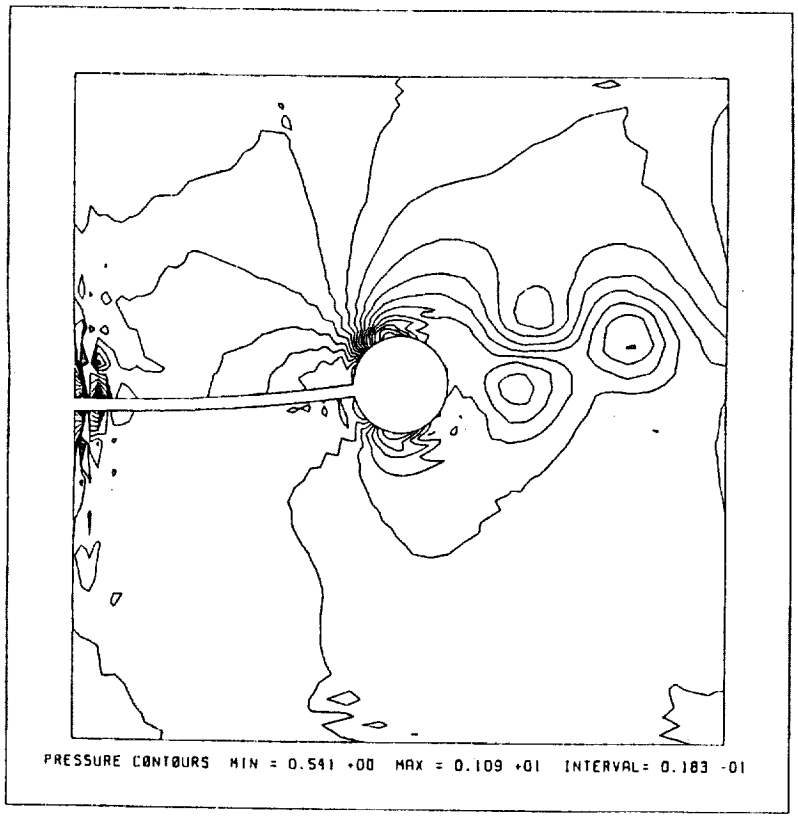


Figure 5.18a: Third deformed configuration of the computational domain for the rigid cylinder, flexible shaft problem using the quasi–steady state method.

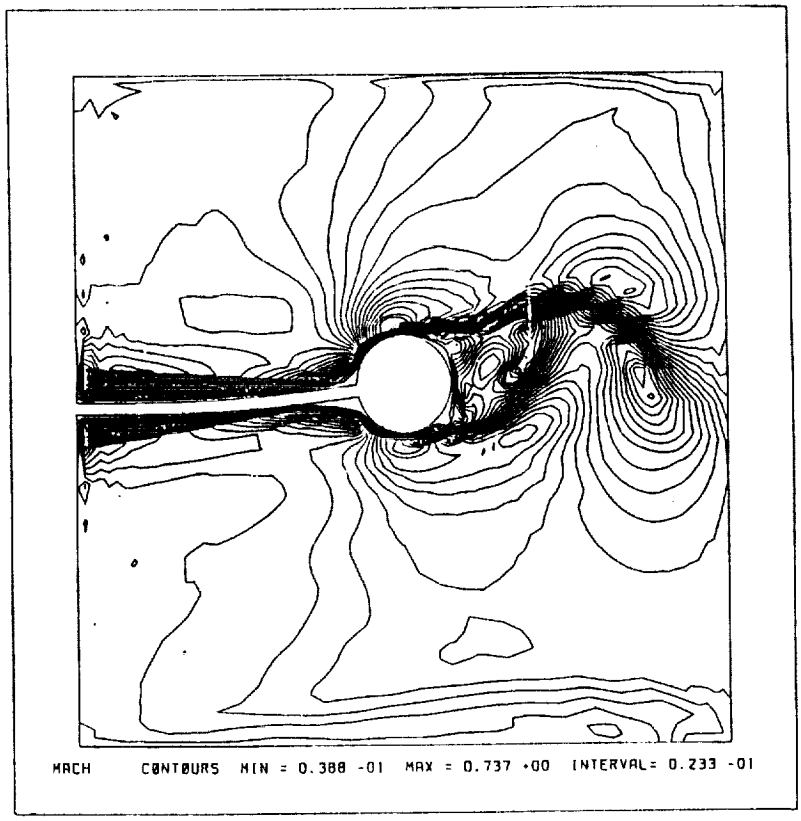


b

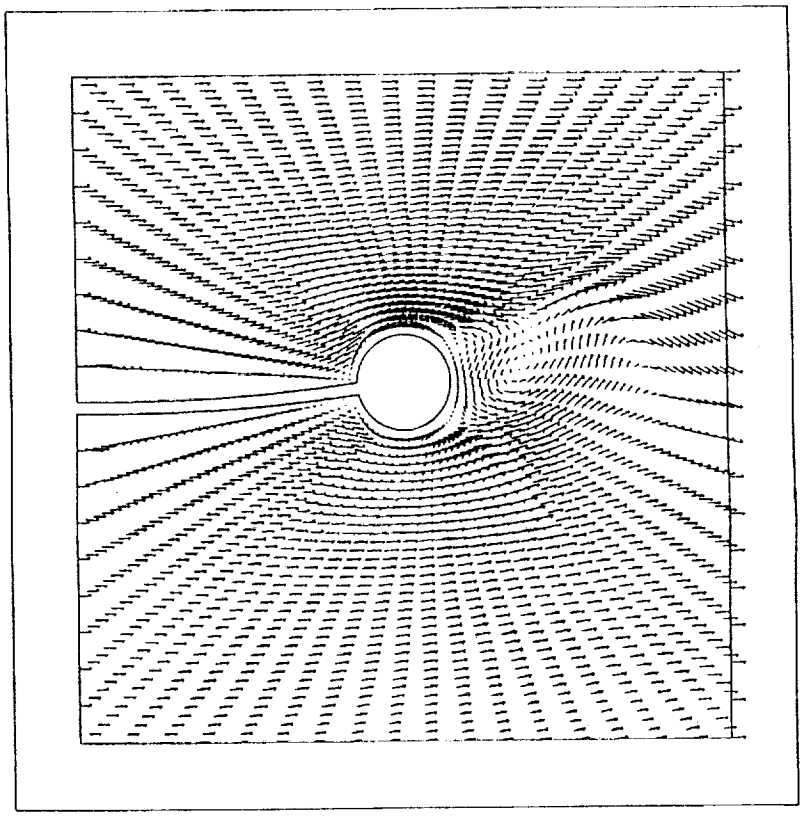


c

Figure 5.18b,c: Results for the rigid cylinder, flexible shaft problem using the quasi-steady state method—third deformed configuration. (b) Density contours, (c) pressure contours.



d



e

Figure 5.18d,e: Results for the rigid cylinder, flexible shaft problem using the quasi-steady state method—third deformed configuration. (d) Mach contours, (e) velocity vectors.

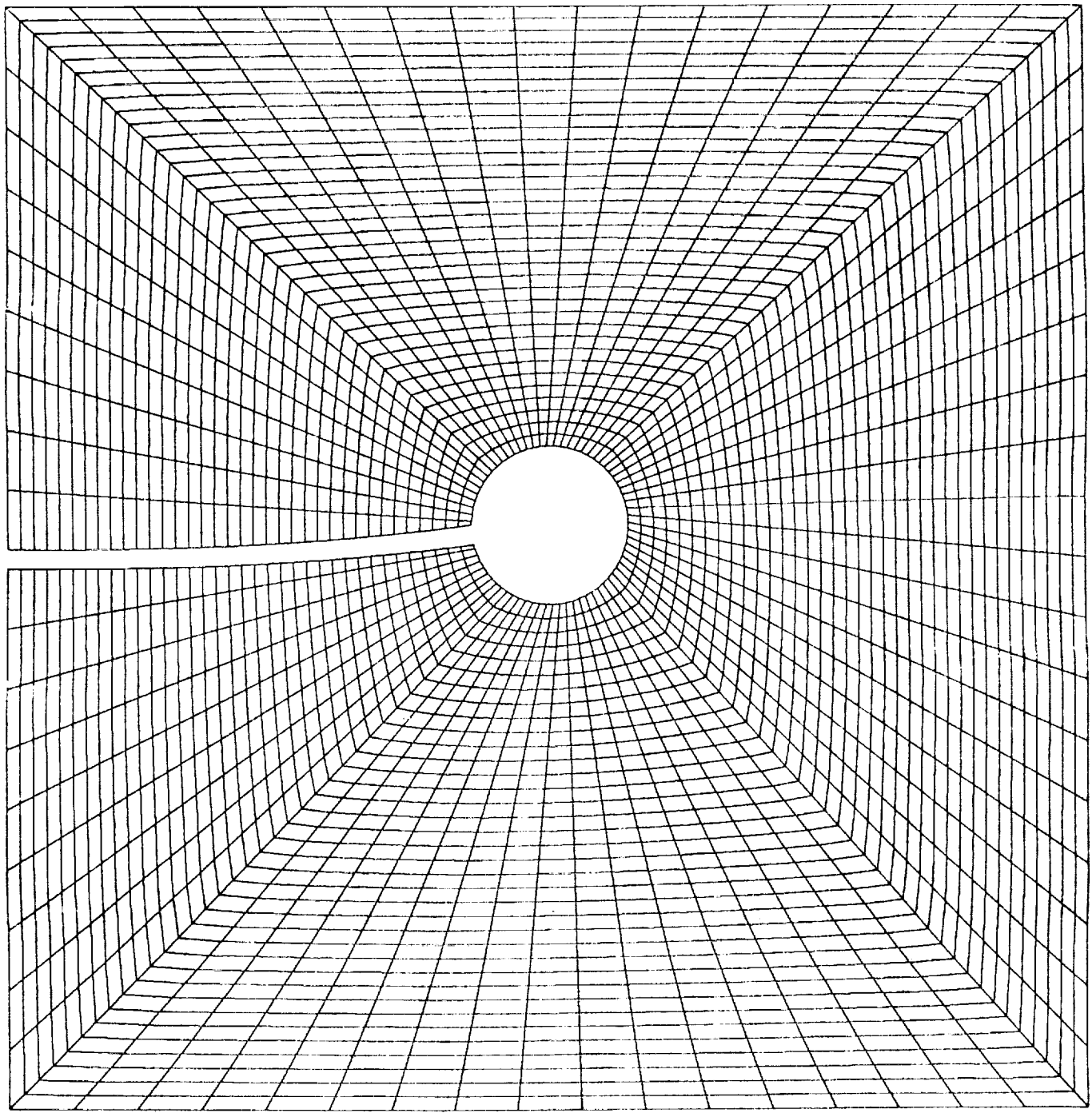
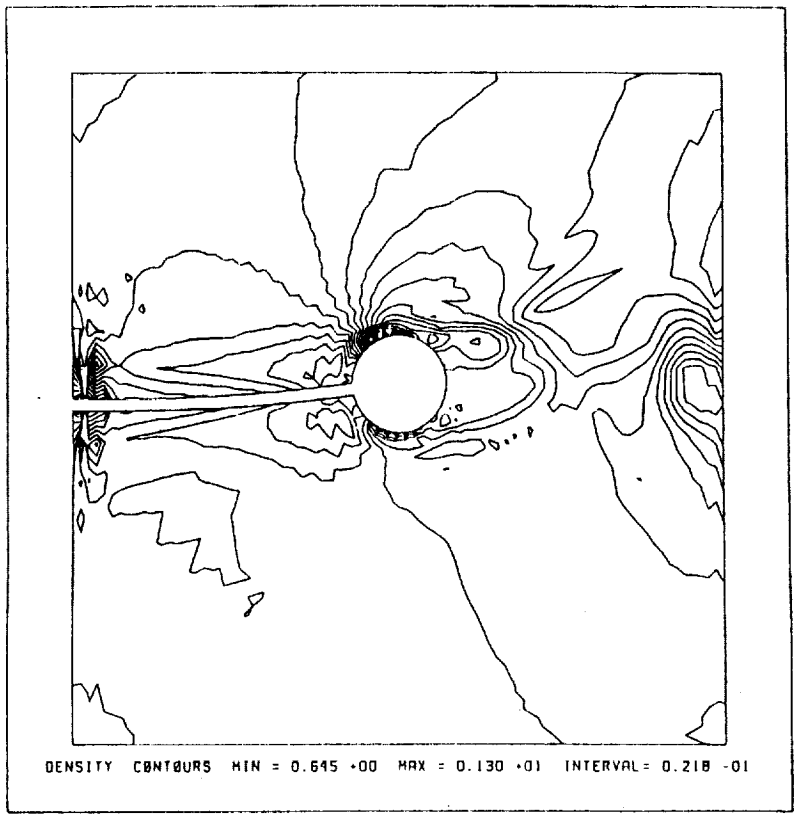
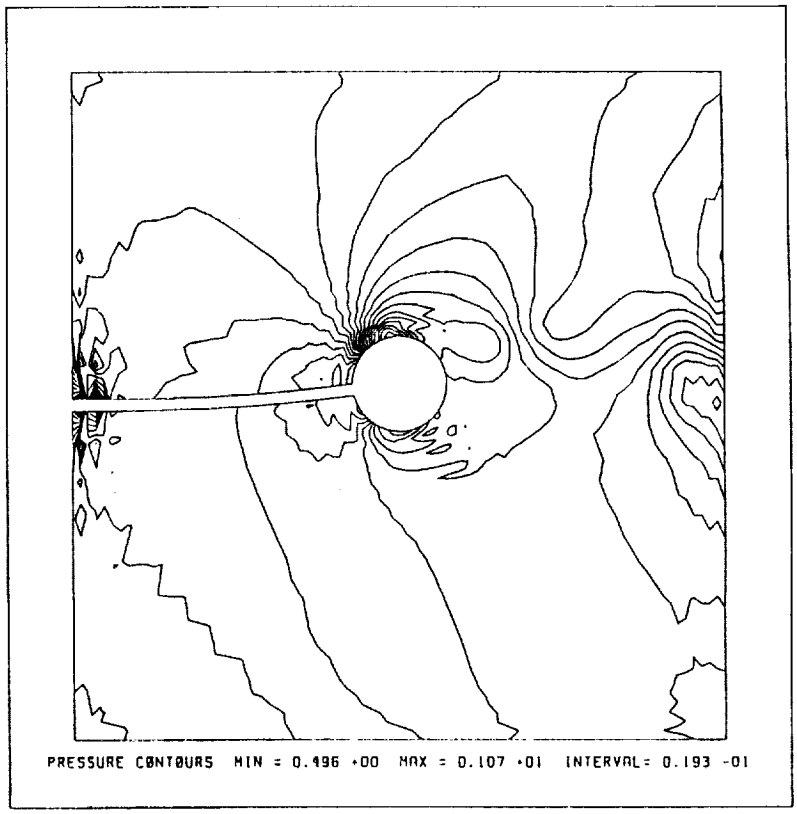


Figure 5.19a: Fourth deformed configuration of the computational domain for the rigid cylinder, flexible shaft problem using the quasi–steady state method.

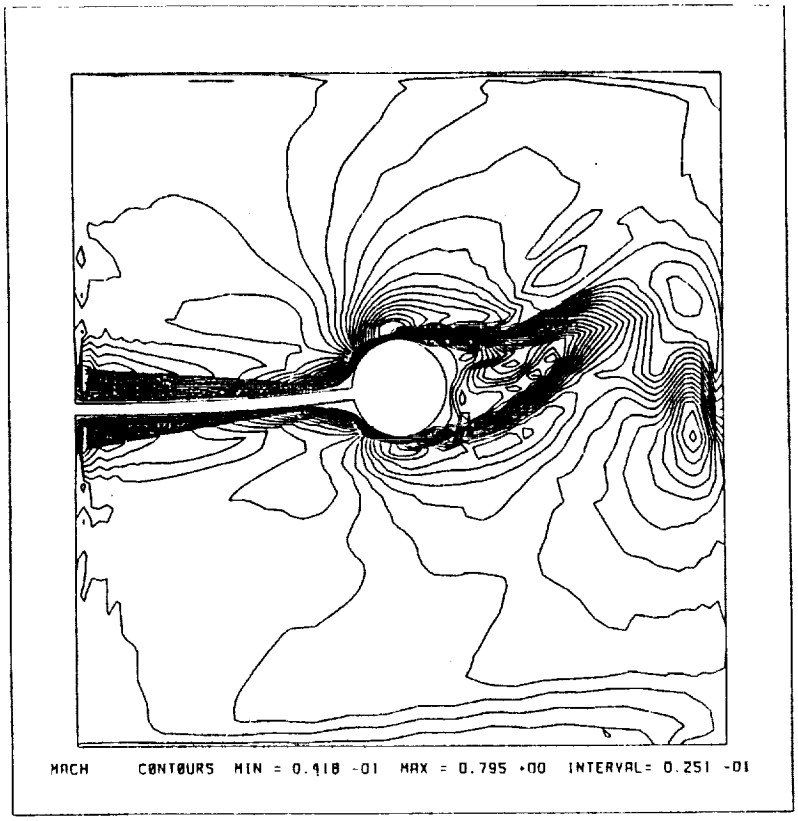


b

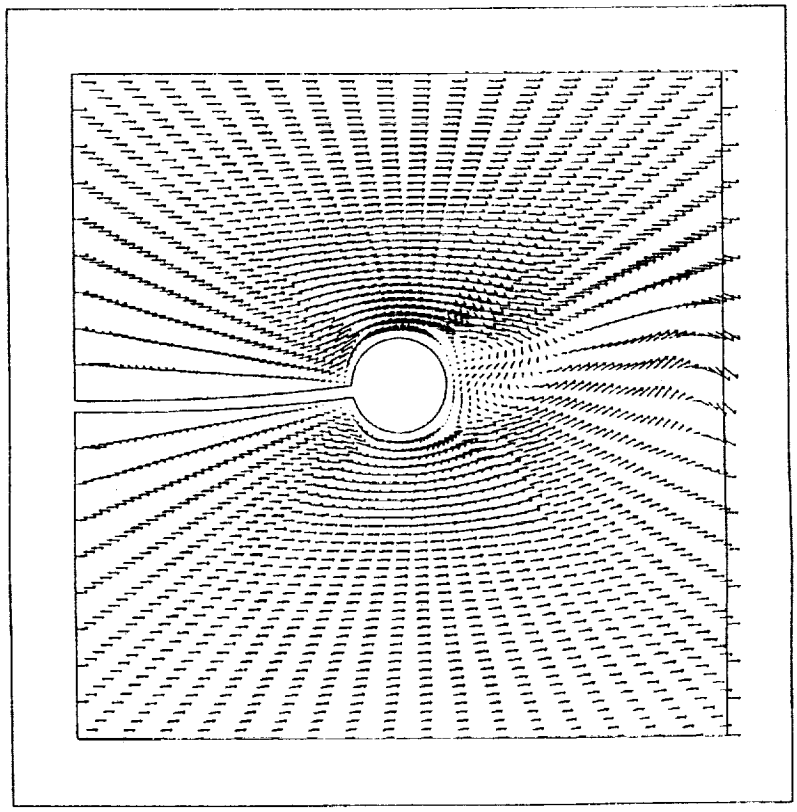


c

Figure 5.19b,c: Results for the rigid cylinder, flexible shaft problem using the quasi–steady state method—fourth deformed configuration. (b) Density contours, (c) pressure contours.



d



e

Figure 5.19d,e: Results for the rigid cylinder, flexible shaft problem using the quasi-steady state method—fourth deformed configuration. (d) Mach contours, (e) velocity vectors.

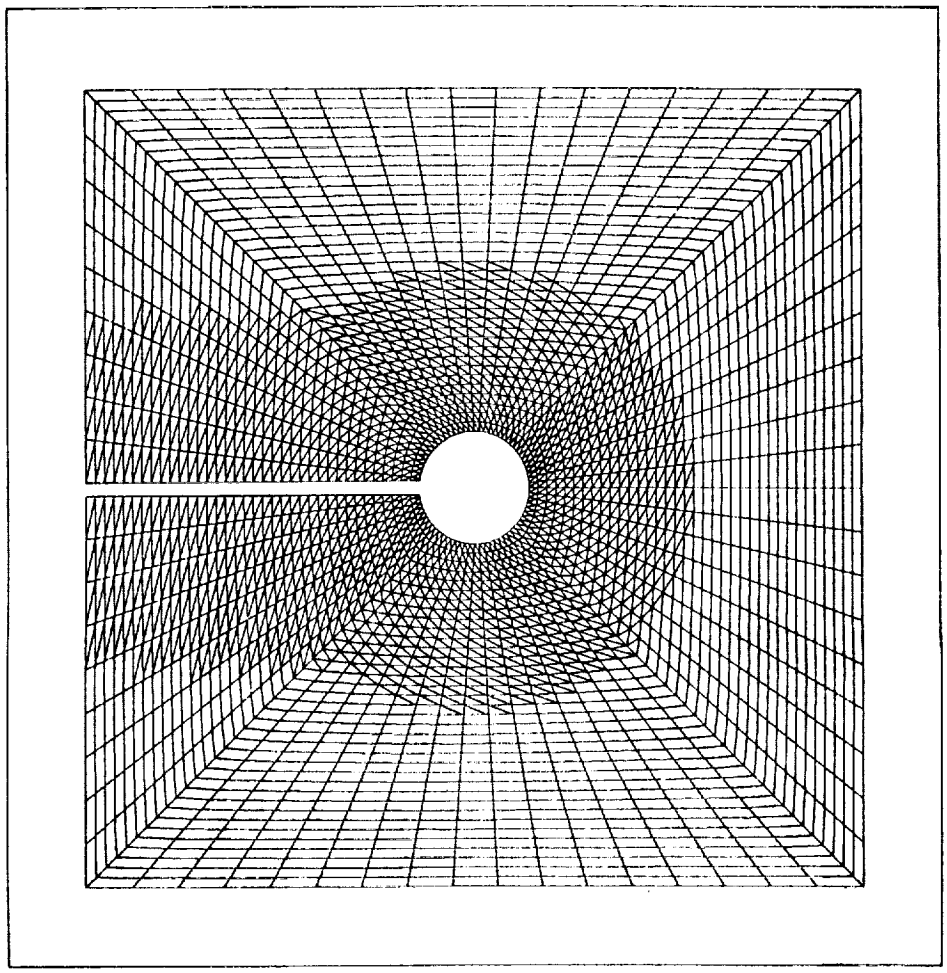


Figure 5.20: Initial grid for the rigid cylinder, flexible shaft problem with remeshing region cross-hatched.

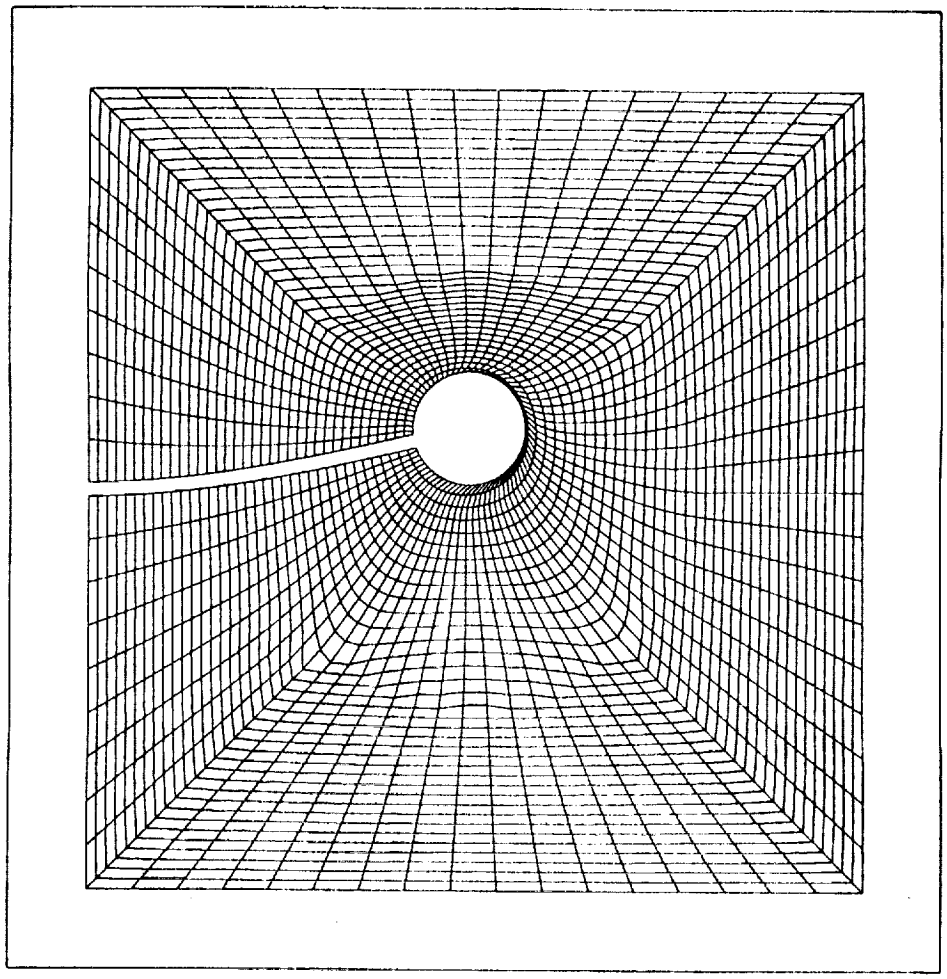


Figure 5.21: Final deformed grid for the rigid cylinder, flexible wall problem using the local remeshing method.

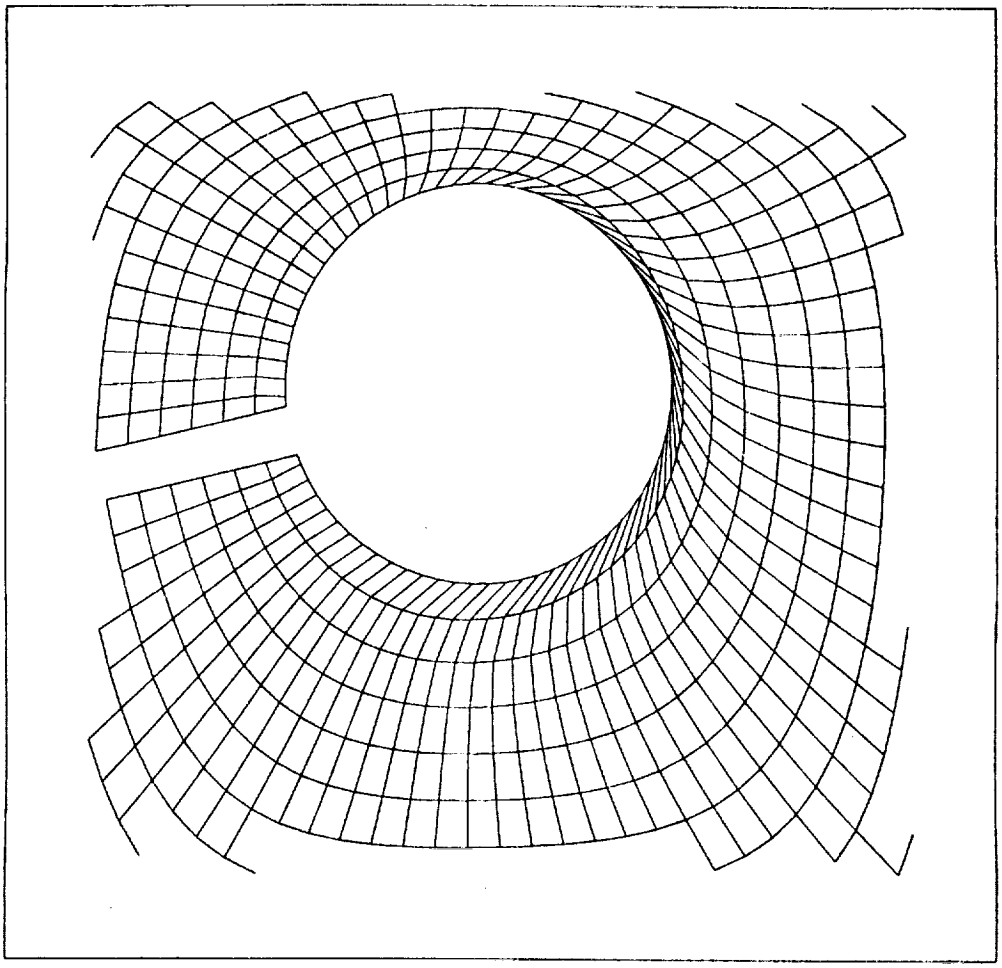


Figure 5.22: Expanded view of the fluid grid in the region of the cylinder showing the development of poor aspect ratios for elements near the tip.

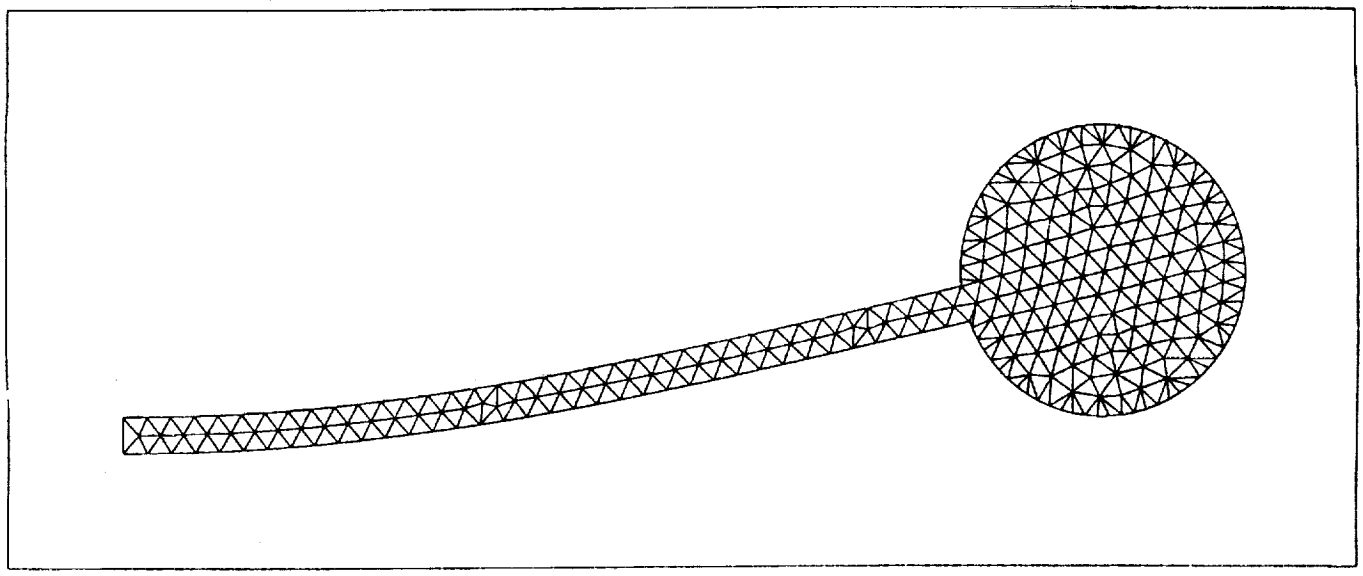


Figure 5.23: Deformed finite element grid used to model the rigid cylinder and flexible shaft with the local remeshing method. This grid was internally generated by the code after specifying boundary points.

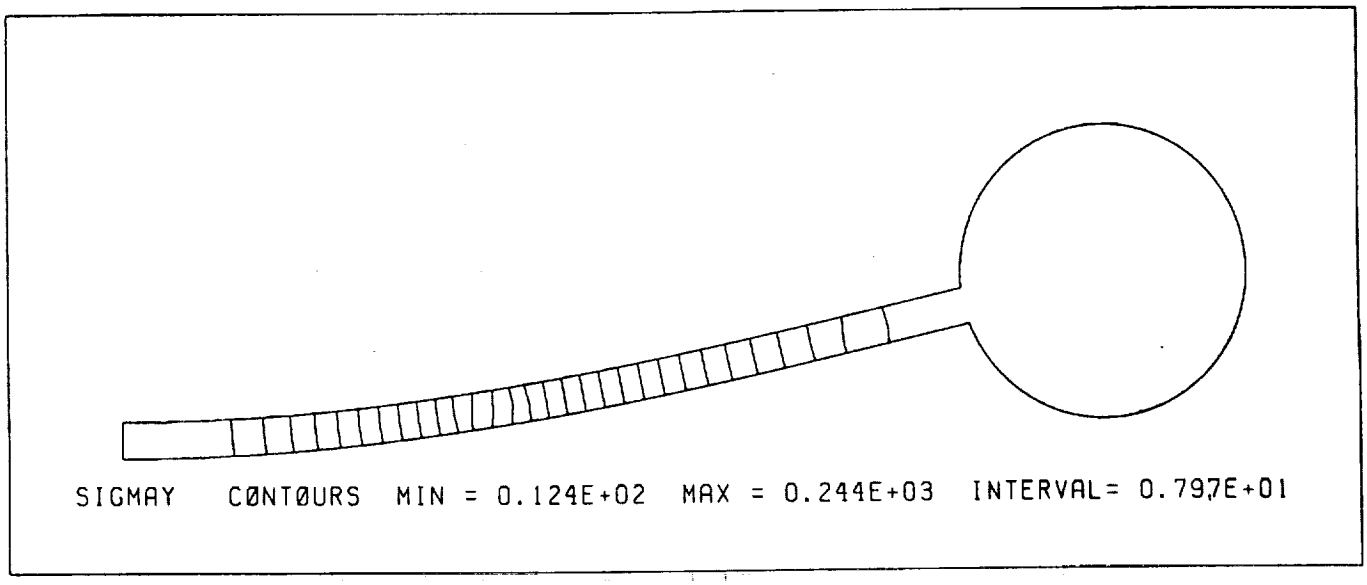


Figure 5.24: Sigma - Y stress contours for the final deformed configuration of the rigid cylinder, flexible shaft problem using the local remeshing method.

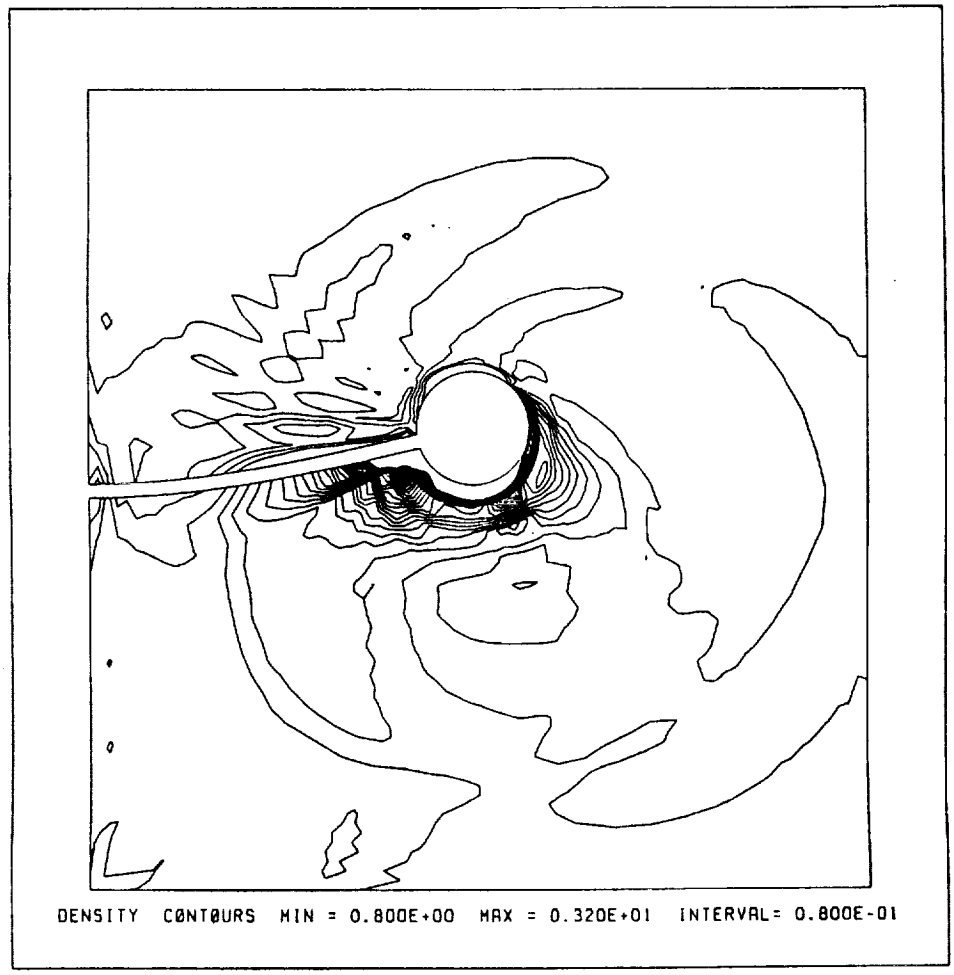


Figure 5.25: Density contours for the final configuration of the rigid cylinder, flexible shaft simulation using the local remeshing method.

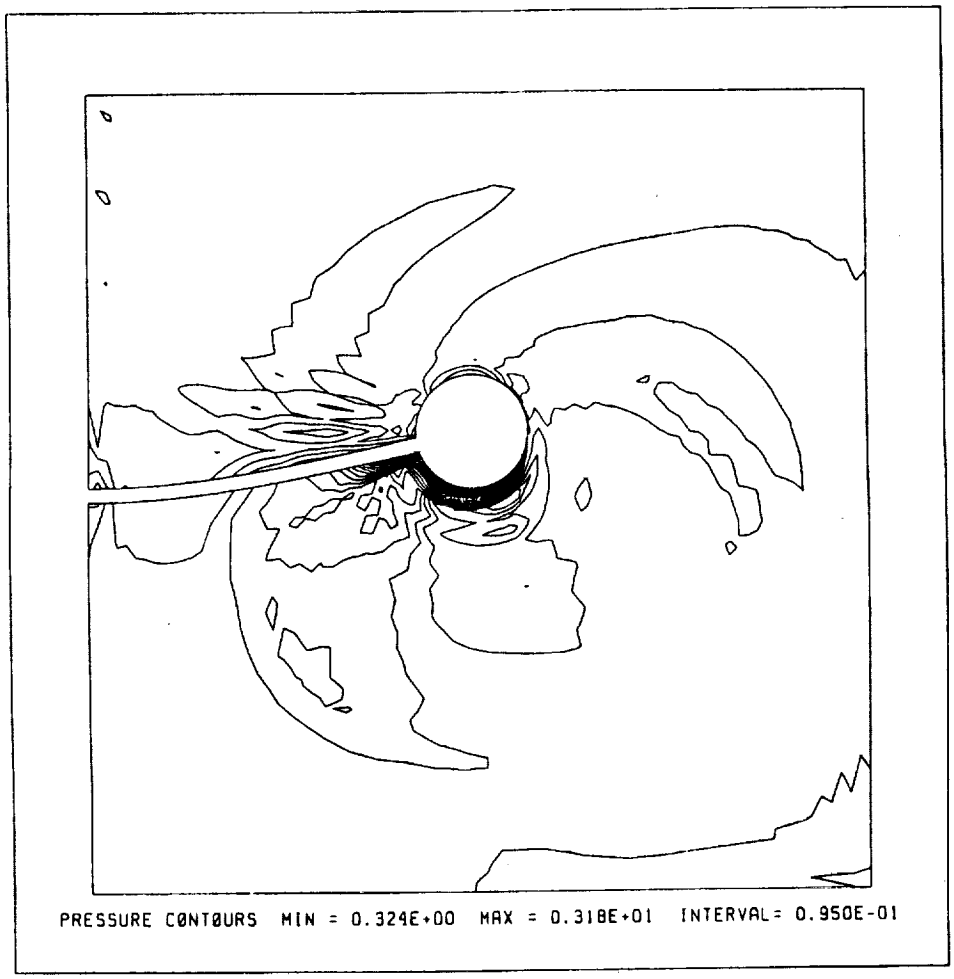


Figure 5.26: Pressure contours for the final configuration of the rigid cylinder, flexible shaft simulation using the local remshing method.

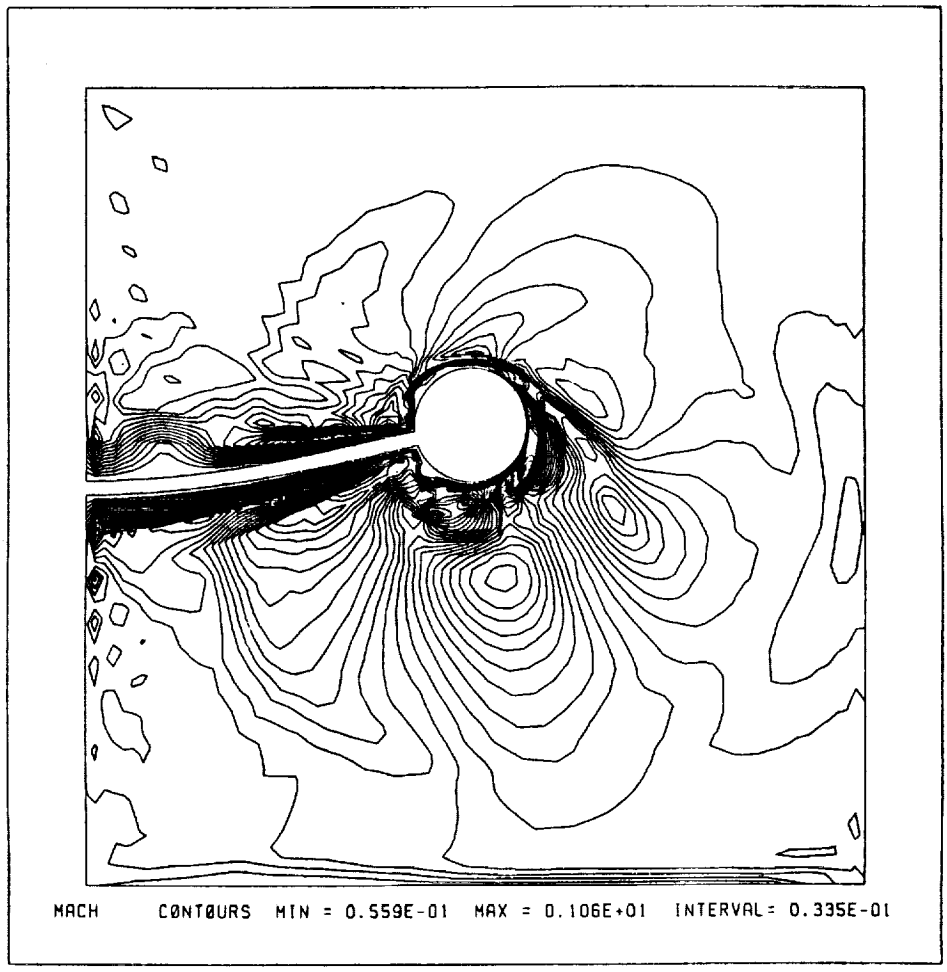


Figure 5.27: Mach contours for the final configuration of the rigid cylinder, flexible shaft simulation using the local remeshing method.

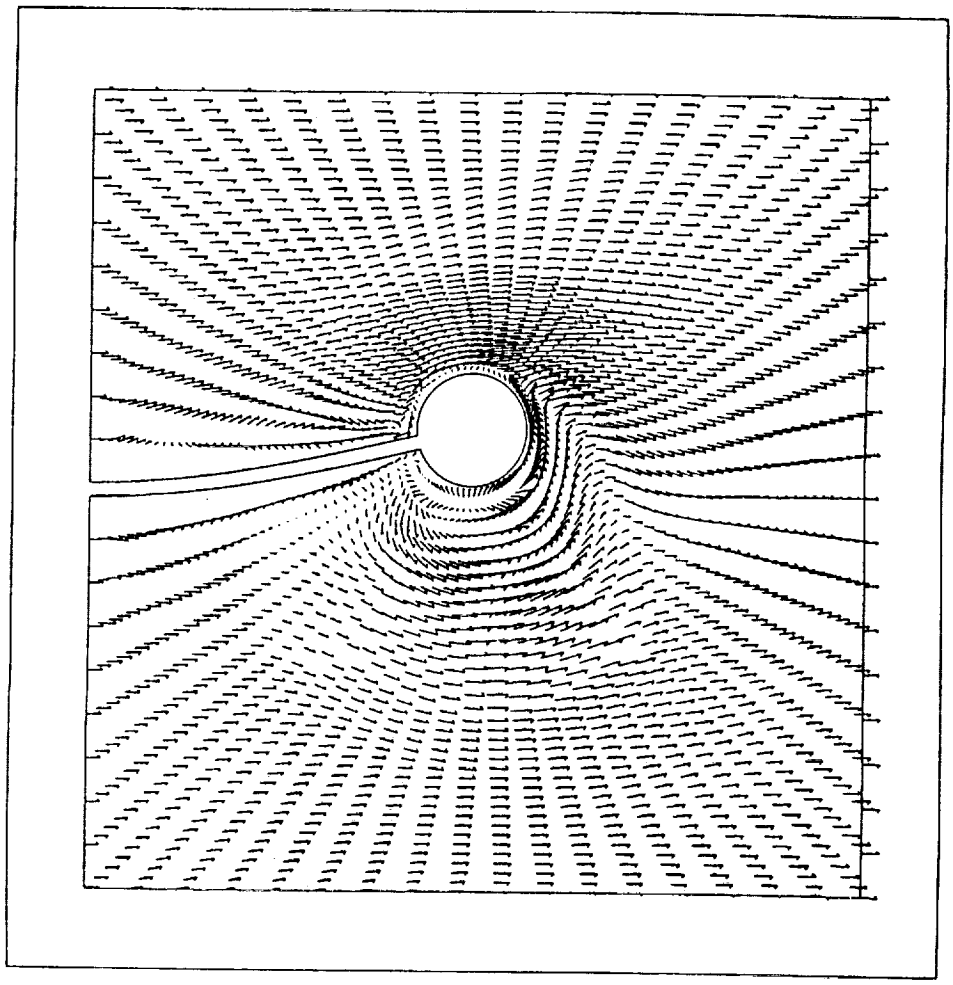
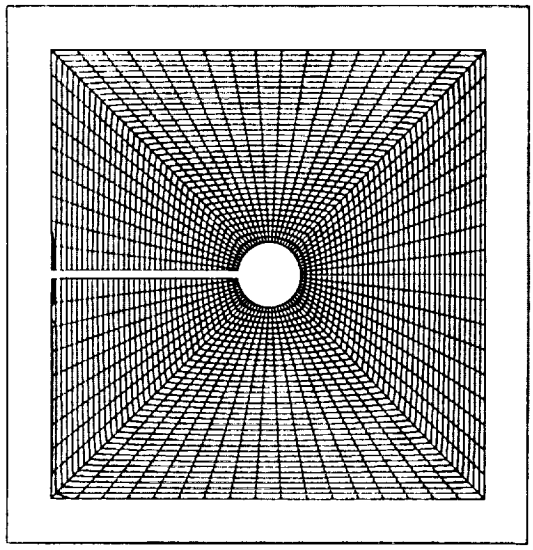
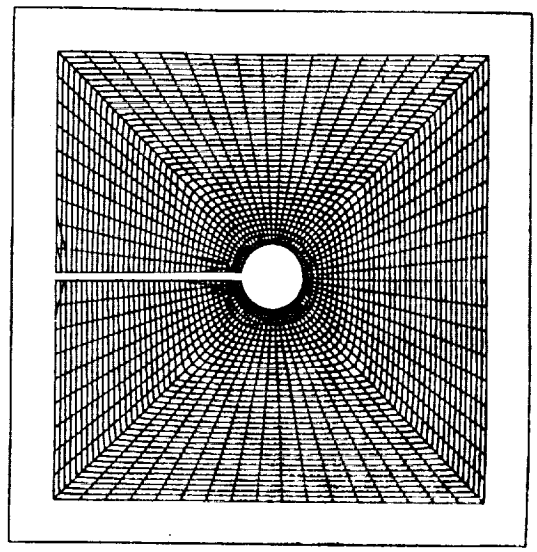


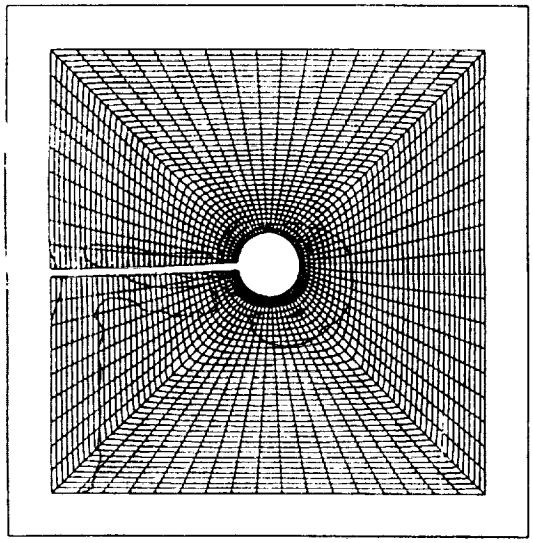
Figure 5.28: Velocity vectors for the final configuration of the rigid cylinder, flexible shaft simulation using the local remeshing method.



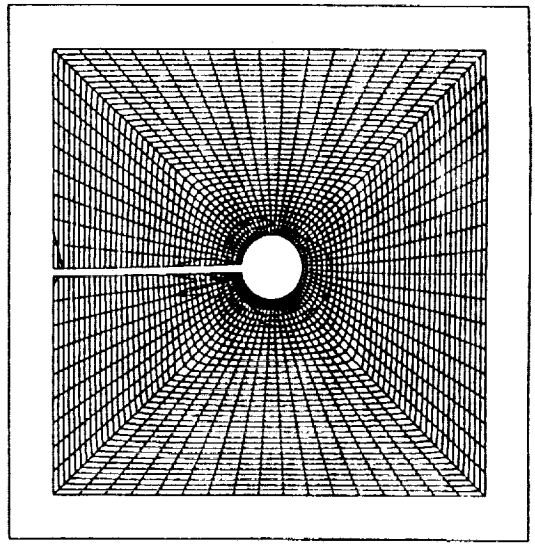
0



20



40



60

Figure 5.29a: Evolution of density contours for the flexible cylinder simulation. Numbers under figures refer to time step.

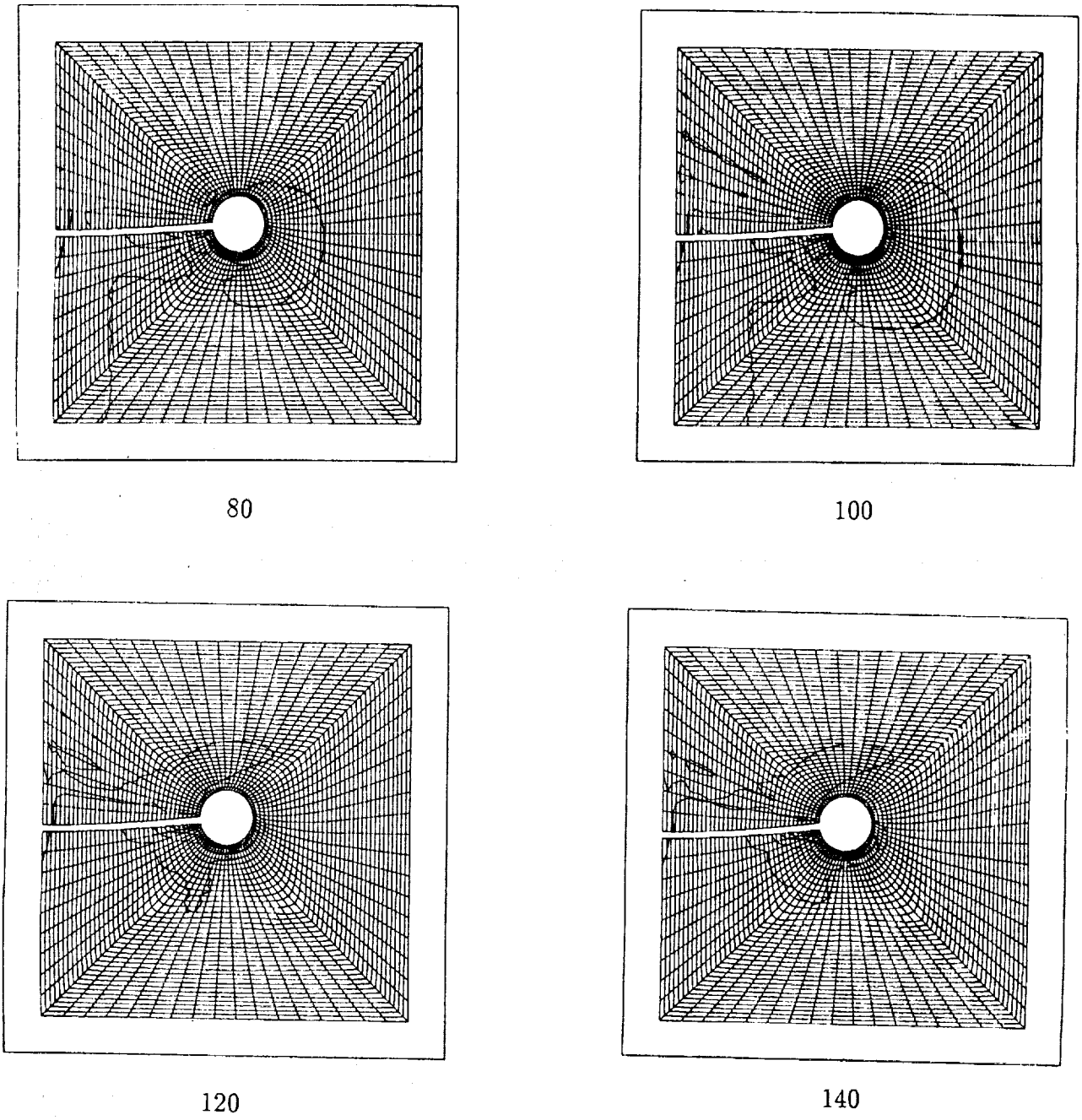
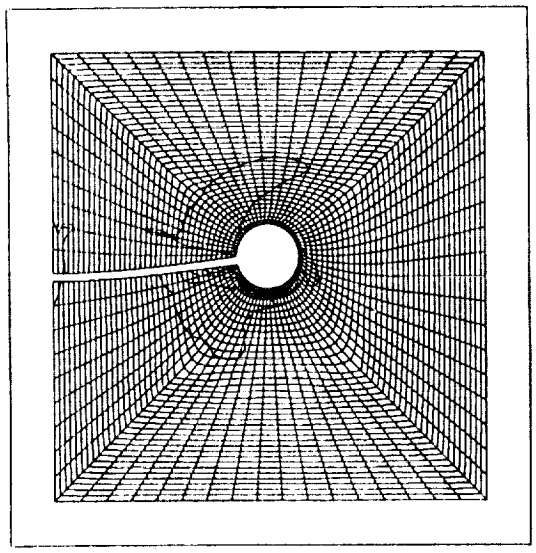
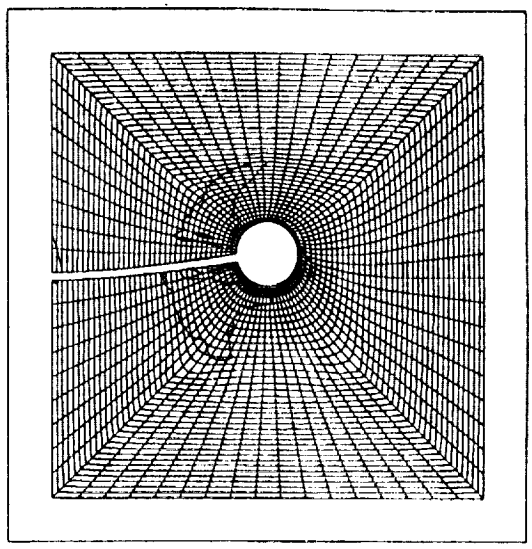


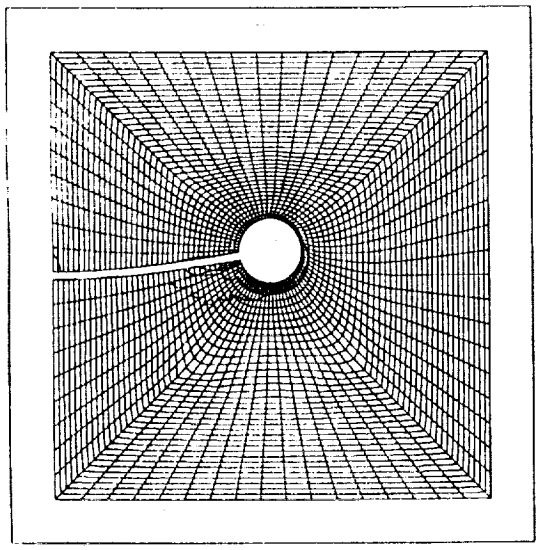
Figure 5.29b: Evolution of density contours for the flexible cylinder simulation. Numbers under figures refer to time step.



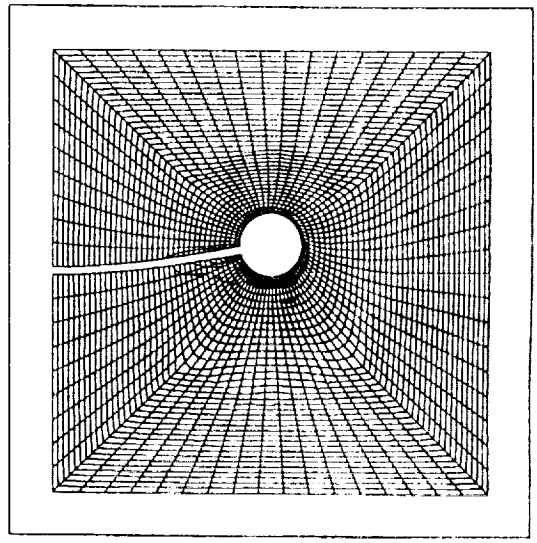
160



180

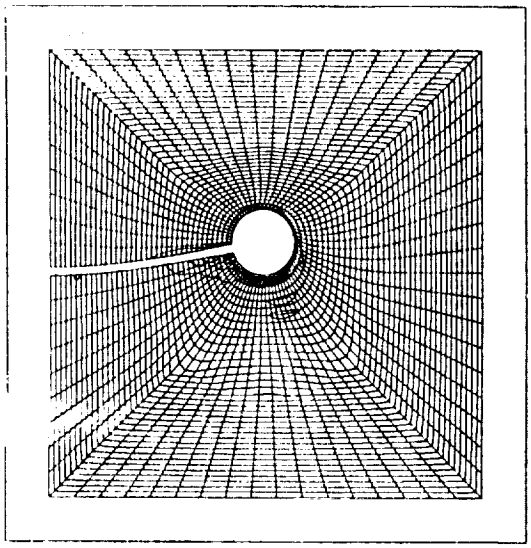


200

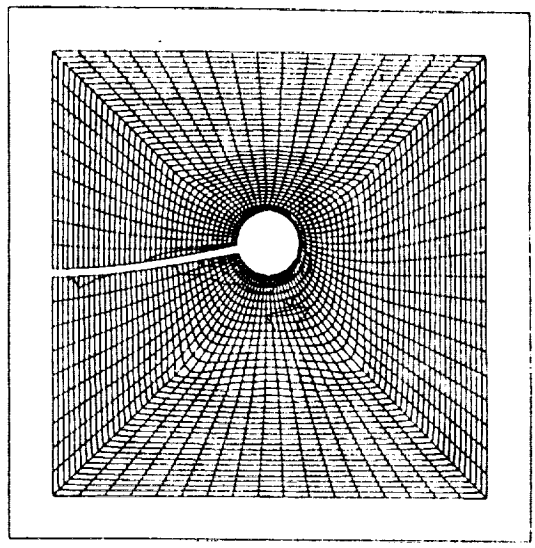


220

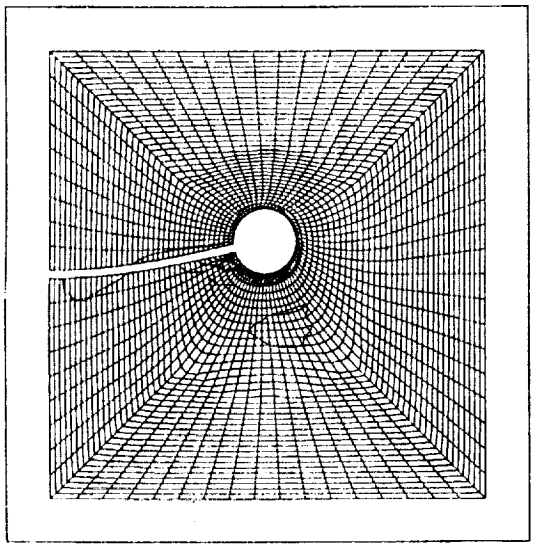
Figure 5.29c: Evolution of density contours for the flexible cylinder simulation. Numbers under figures refer to time step.



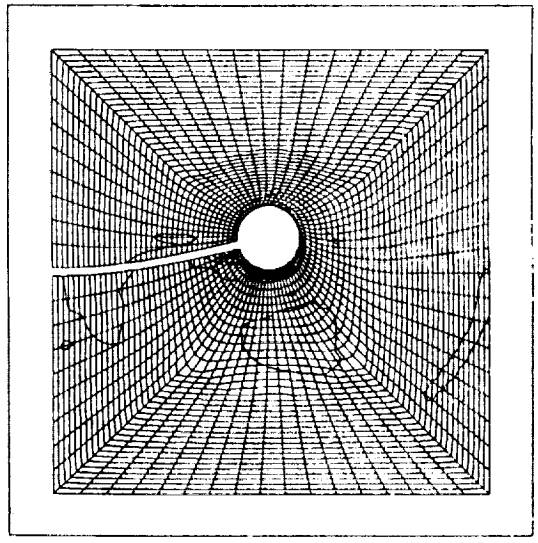
240



260

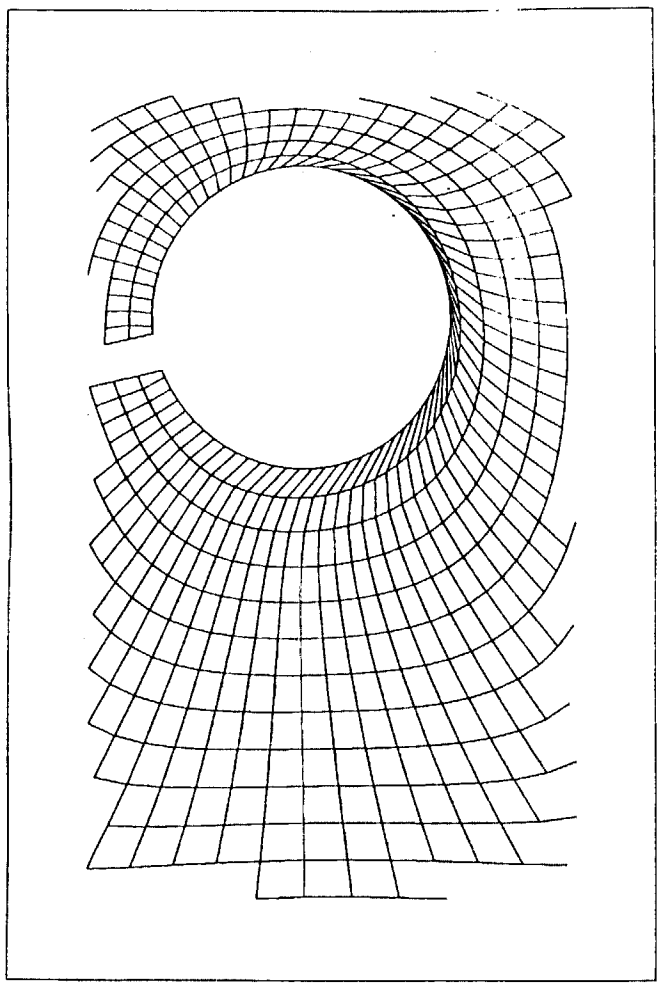


280



300

Figure 5.29d: Evolution of density contours for the flexible cylinder simulation. Numbers under figures refer to time step.



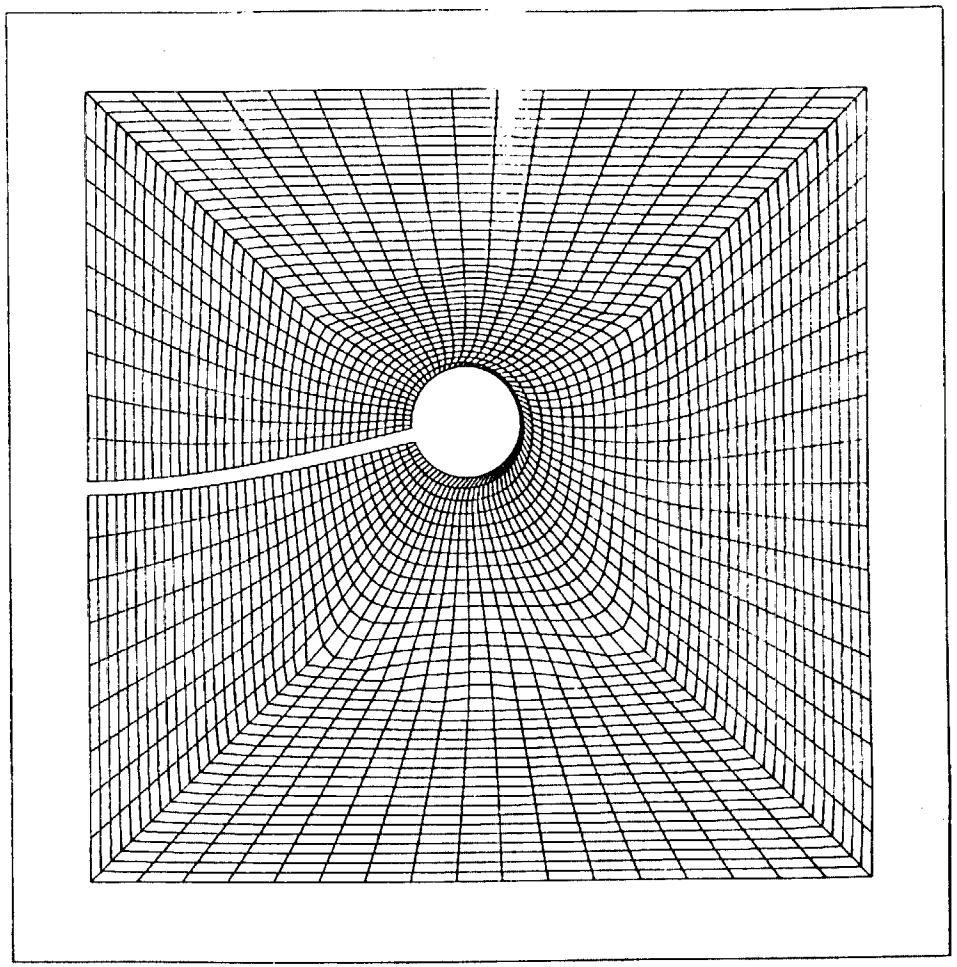
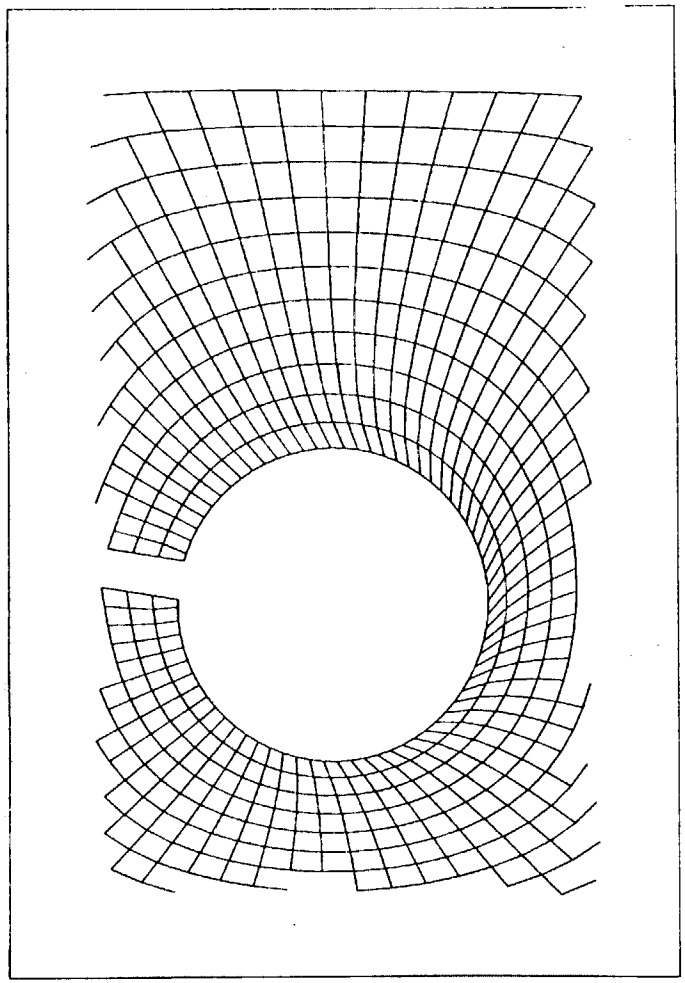


Figure 5.30: Parameter study results: Case 1. Orthogonality = 0.25, Smoothness = 0.75.
(a) Enlarged region around cylinder, (b) computational domain.



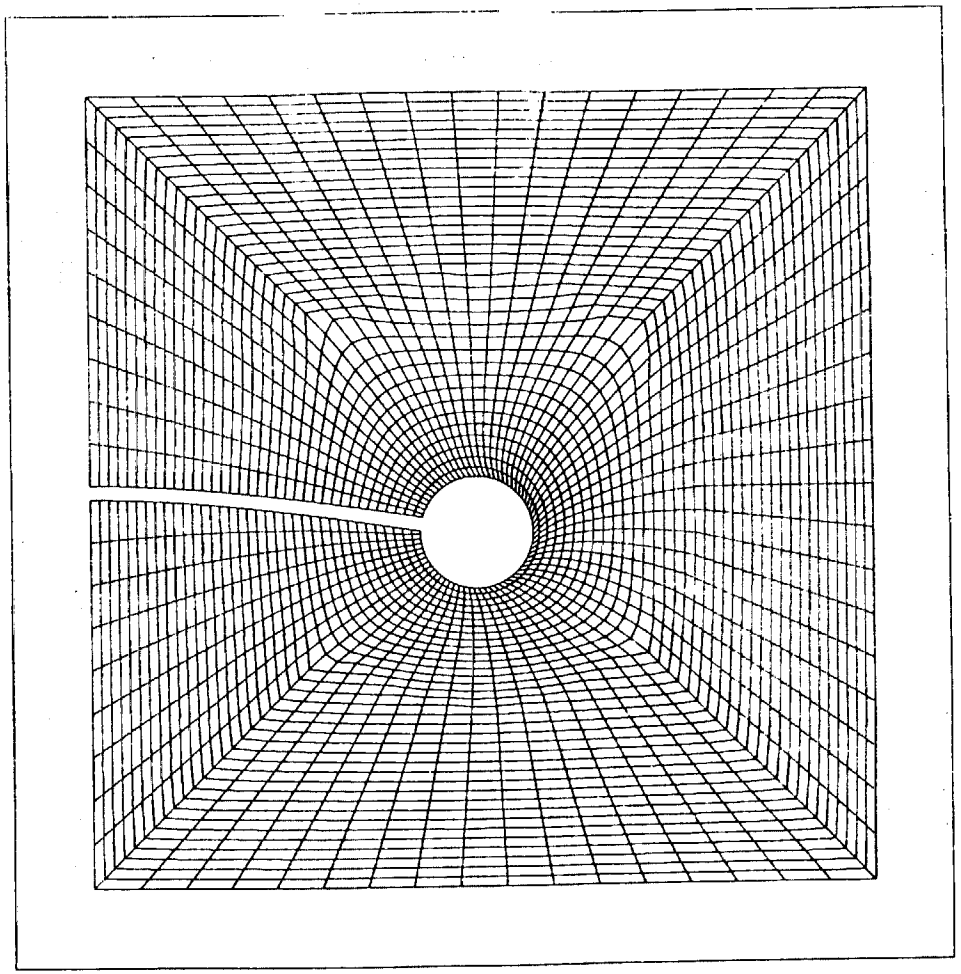
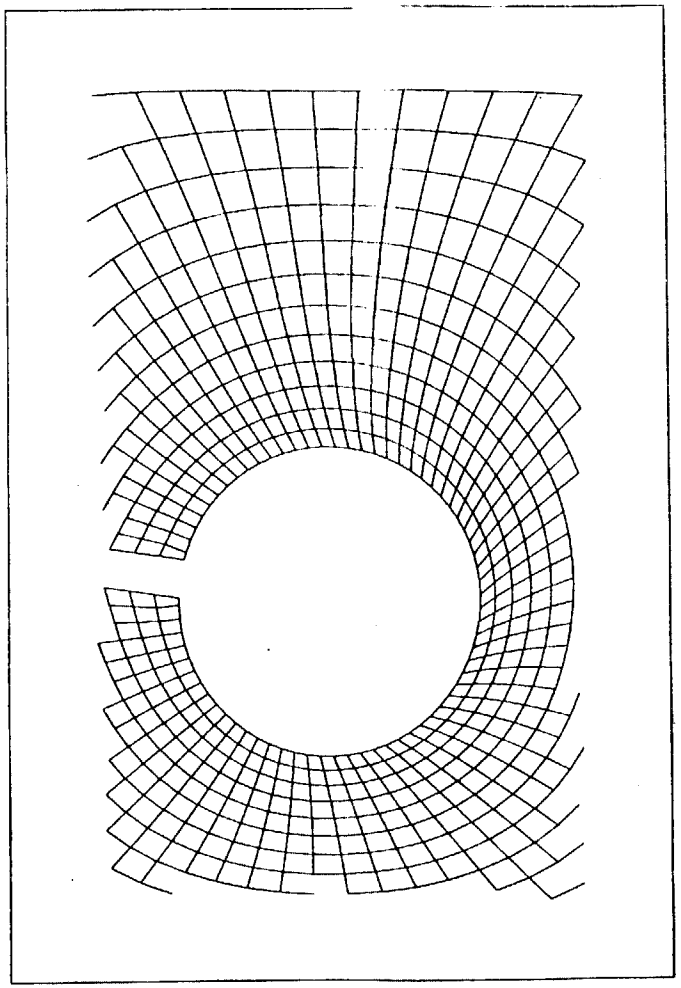


Figure 5.31: Parameter study results: Case 2. Orthogonality = 0.40, Smoothness = 0.60.
(a) Enlarged region around cylinder, (b) computational domain.



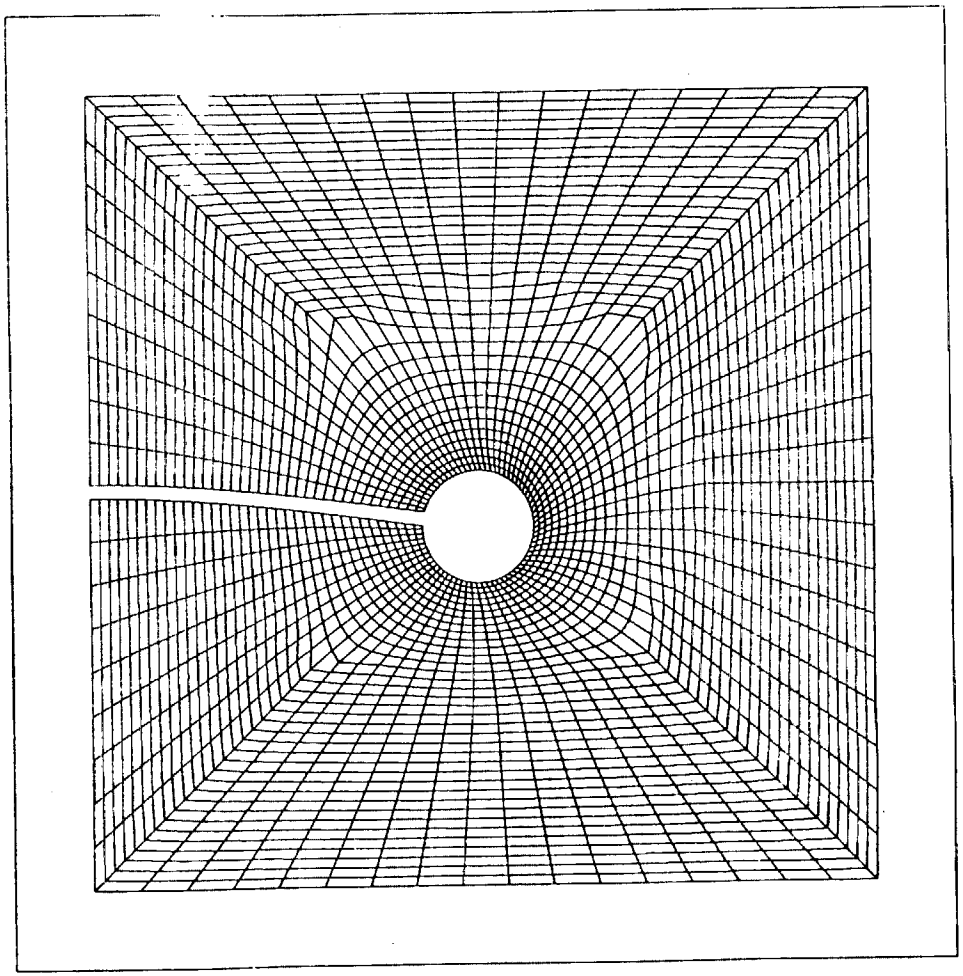
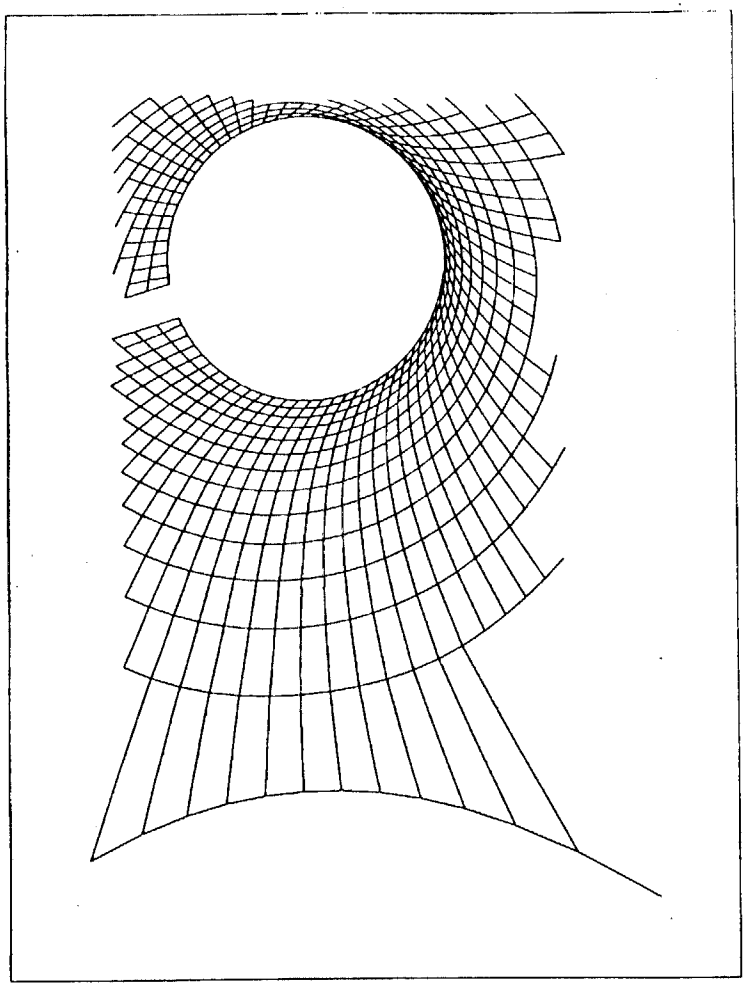


Figure 5.32: Parameter study results: Case 3. Orthogonality = 0.60, Smoothness = 0.40.
(a) Enlarged region around cylinder, (b) computational domain.



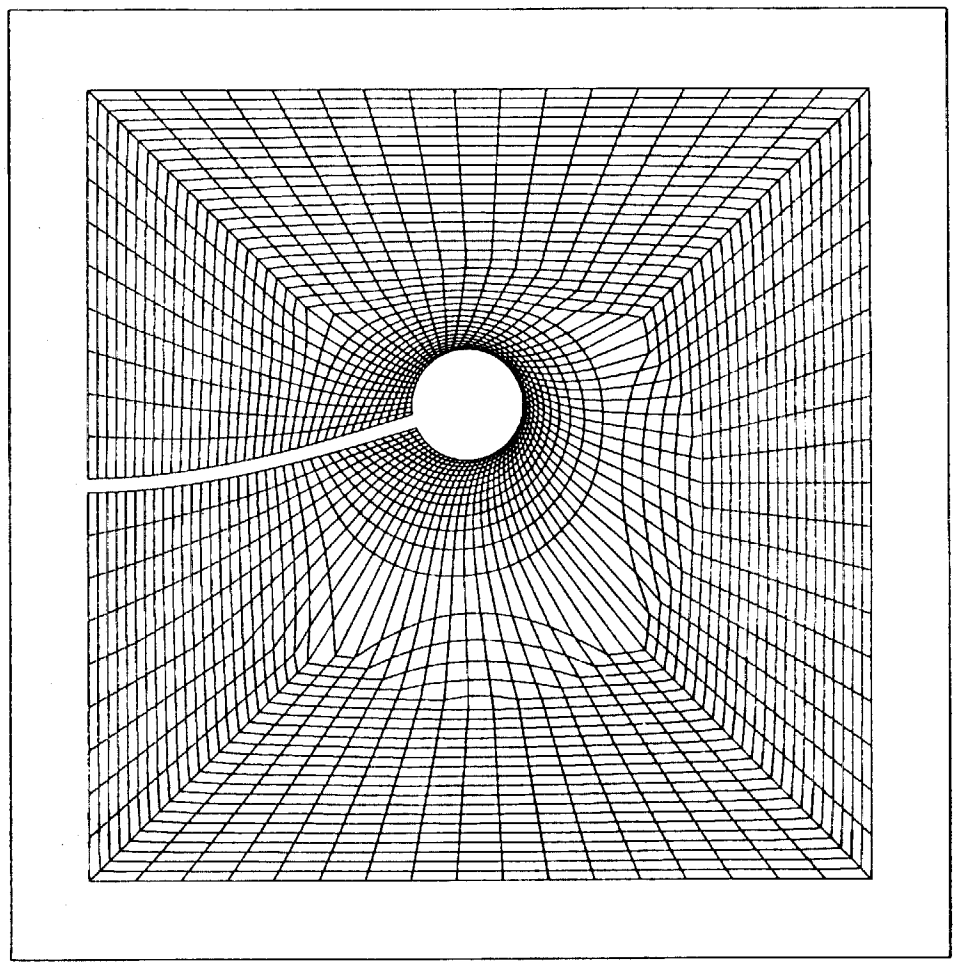
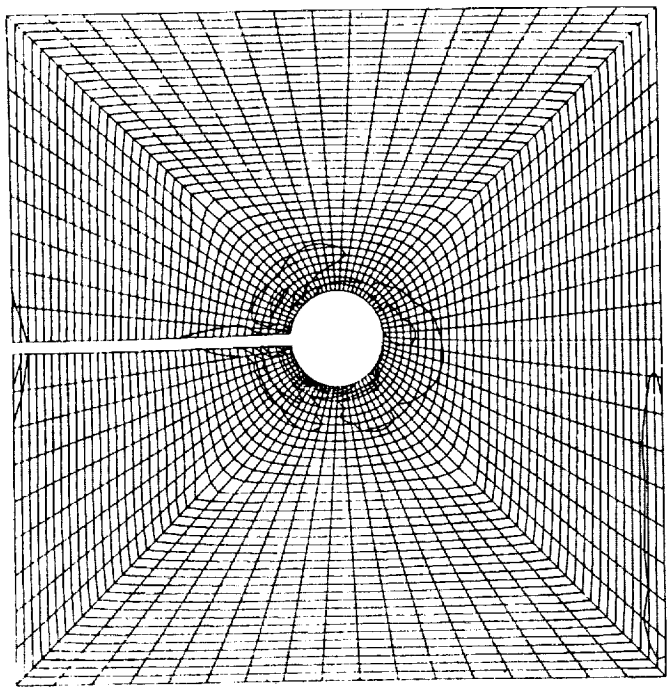
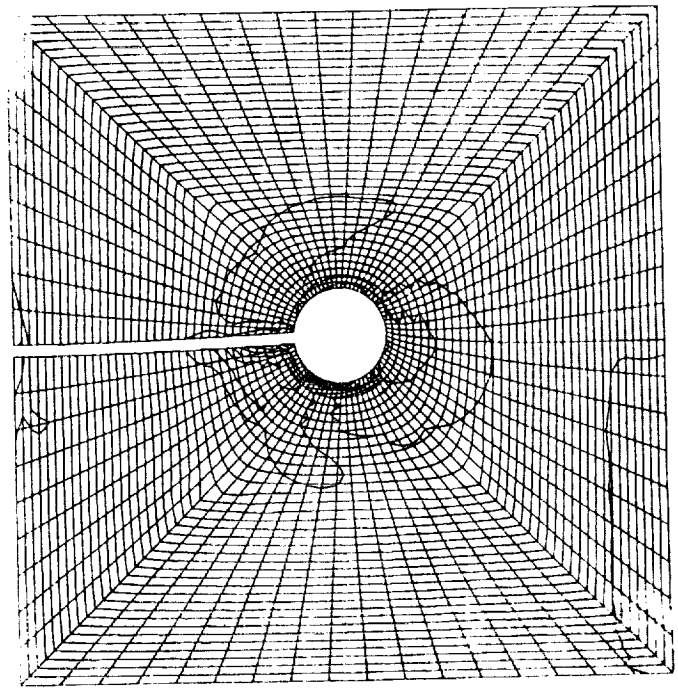


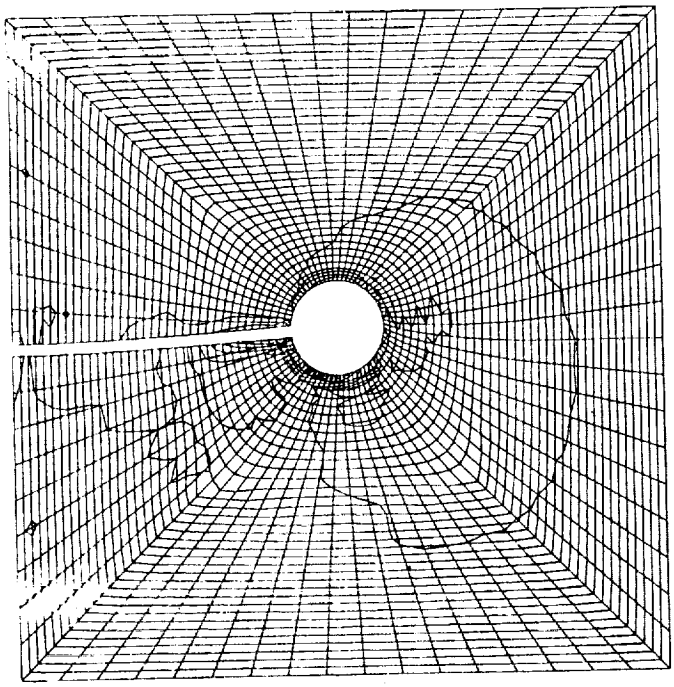
Figure 5.33: Parameter study results: Case 4. Orthogonality = 0.80, Smoothness = 0.20.
(a) Enlarged region around cylinder, (b) computational domain.



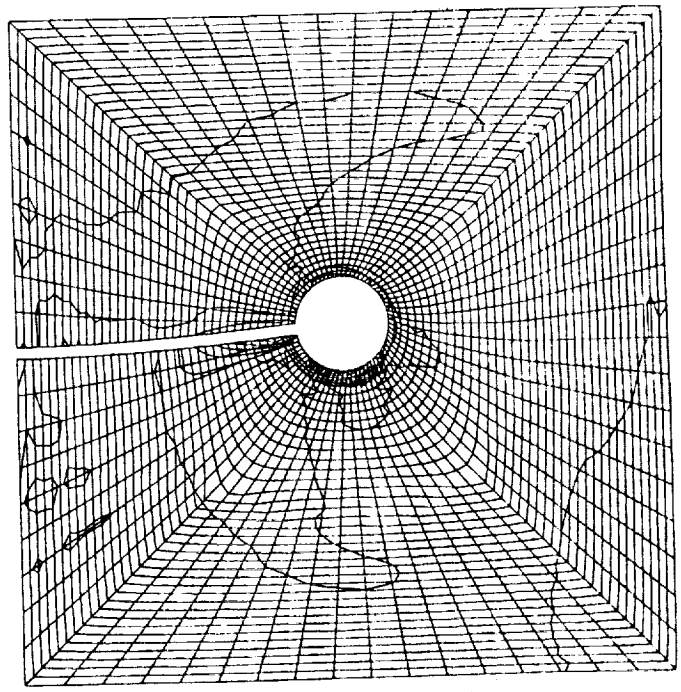
50



100

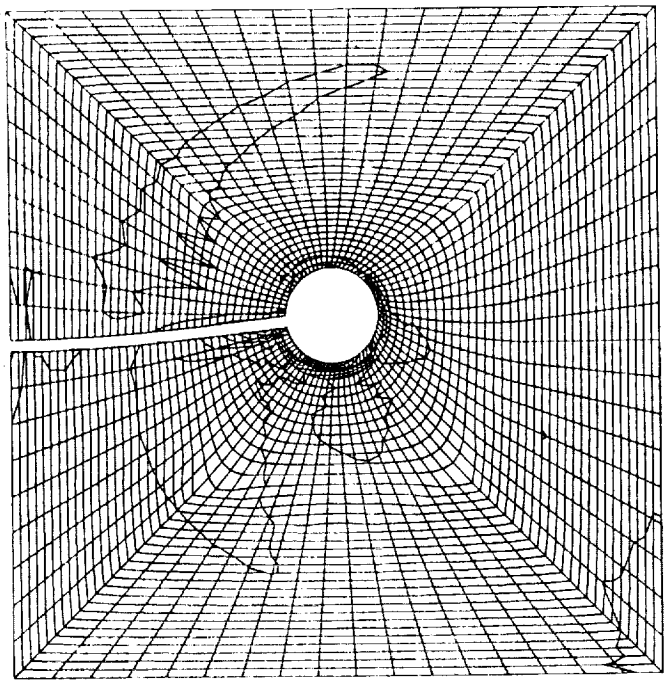


150

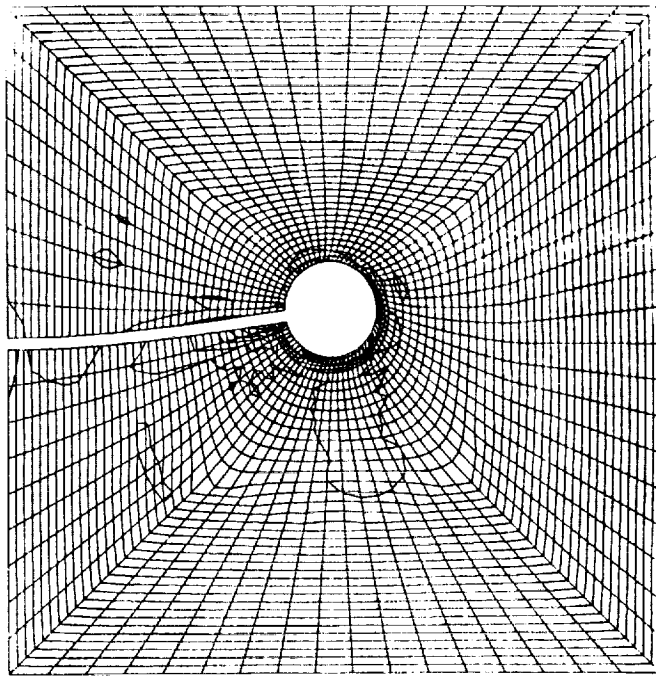


200

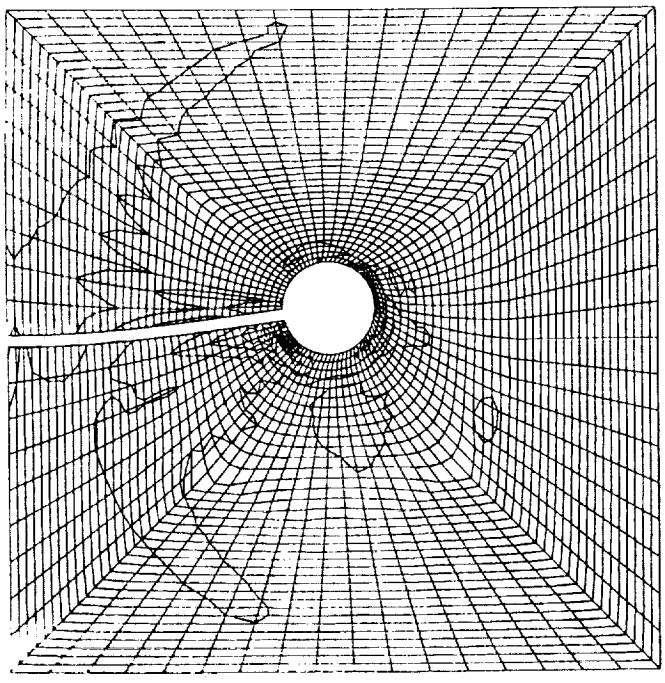
Figure 5.34a: Evolution of density contours for Case 2 parameter study: Orthogonality = 0.40, Smoothness = 0.60. Numbers under figures refer to time step.



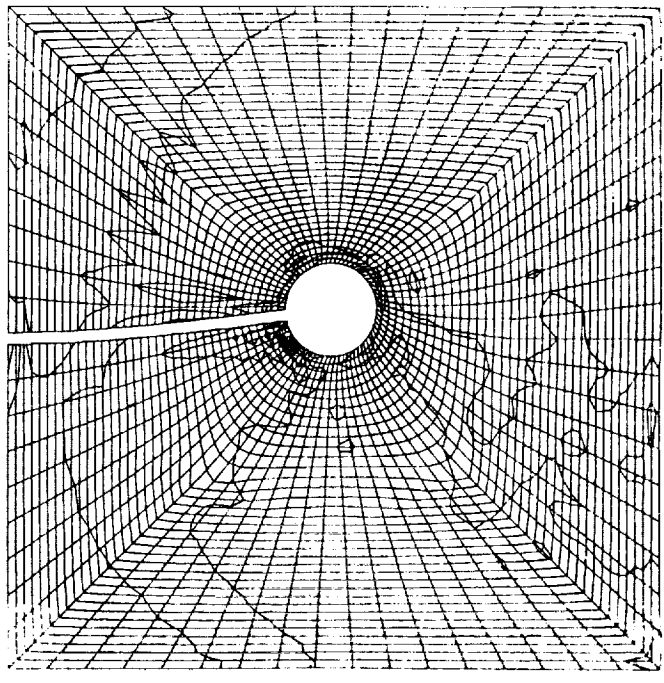
250



300

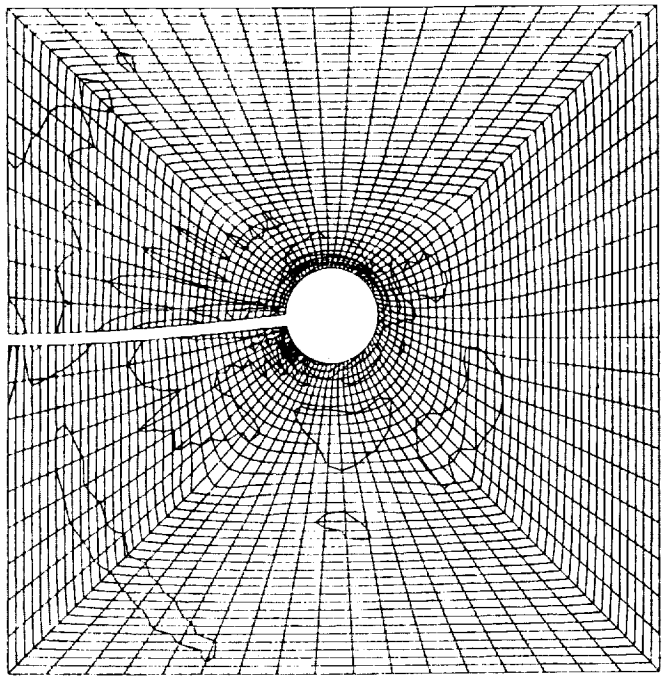


350

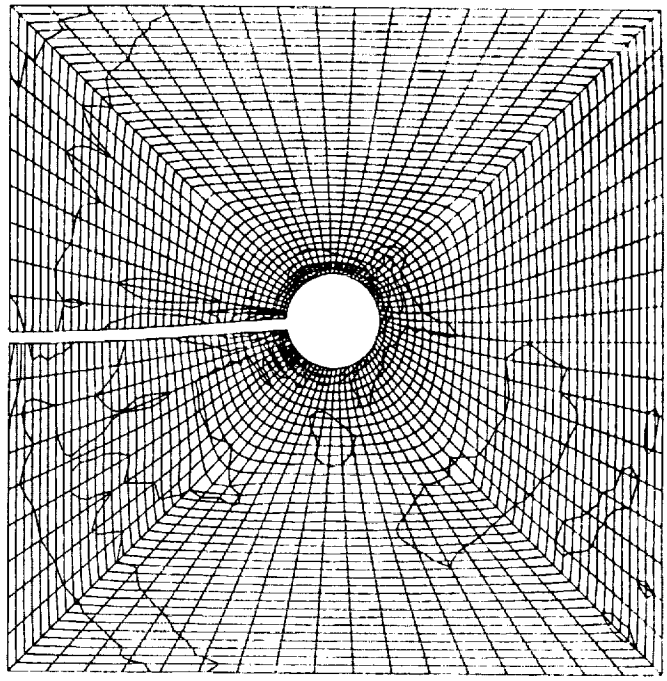


400

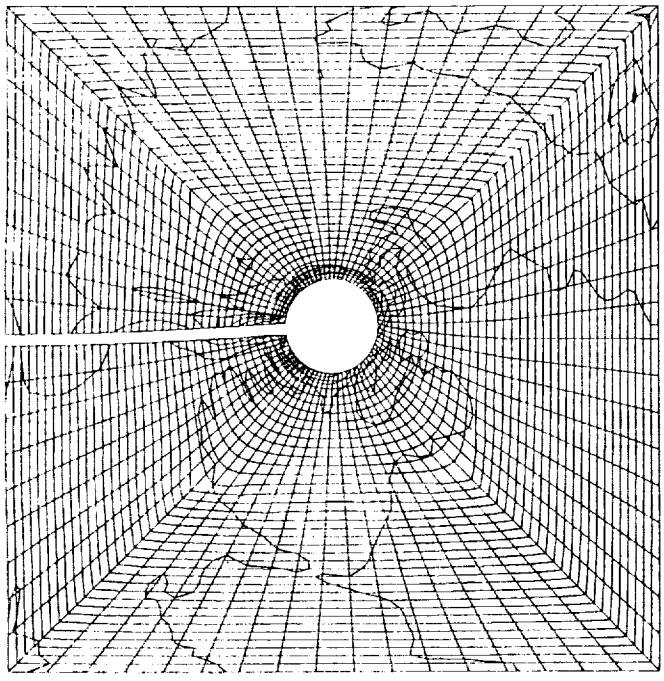
Figure 5.34b: Evolution of density contours for Case 2 parameter study: Orthogonality = 0.40, Smoothness = 0.60. Numbers under figures refer to time step.



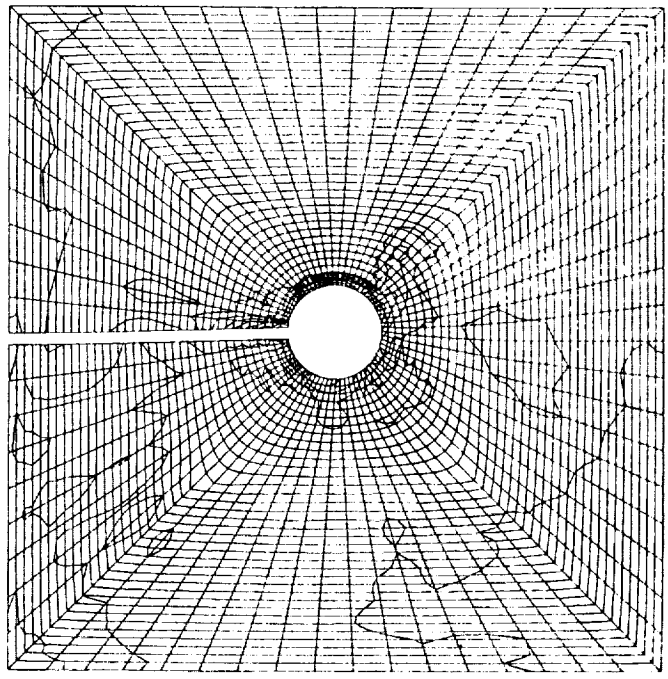
450



500

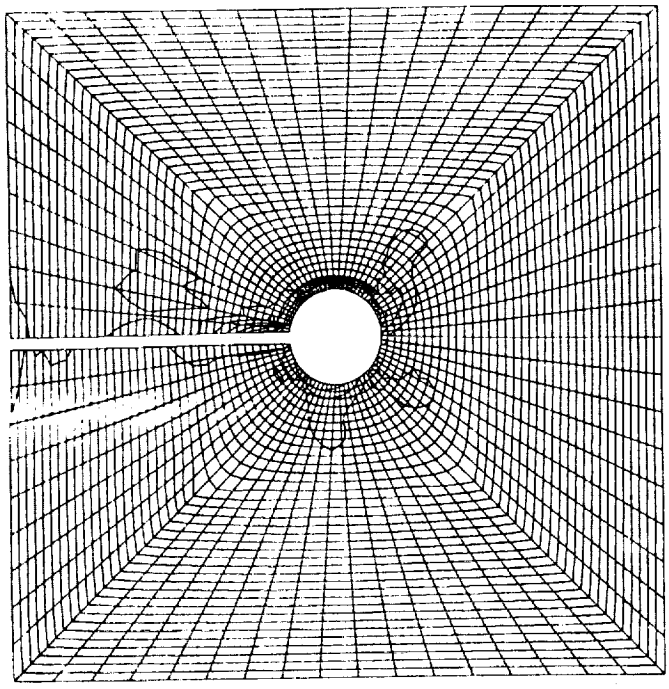


550

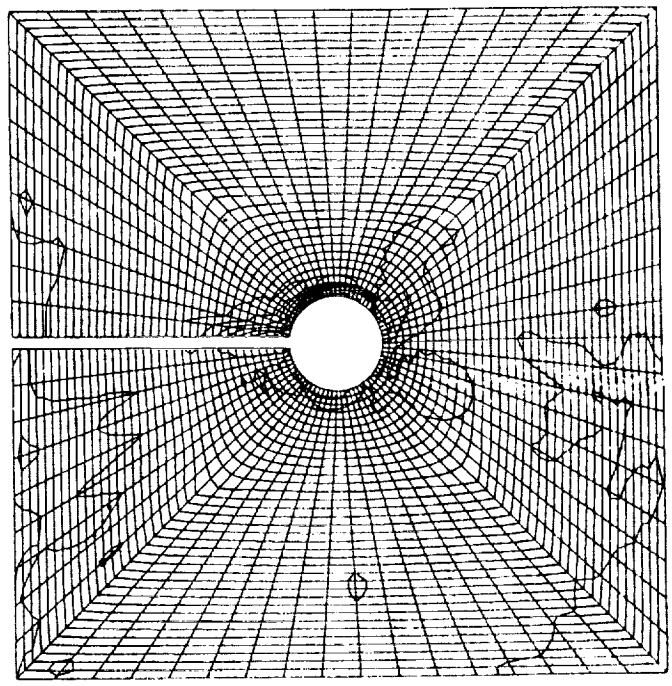


600

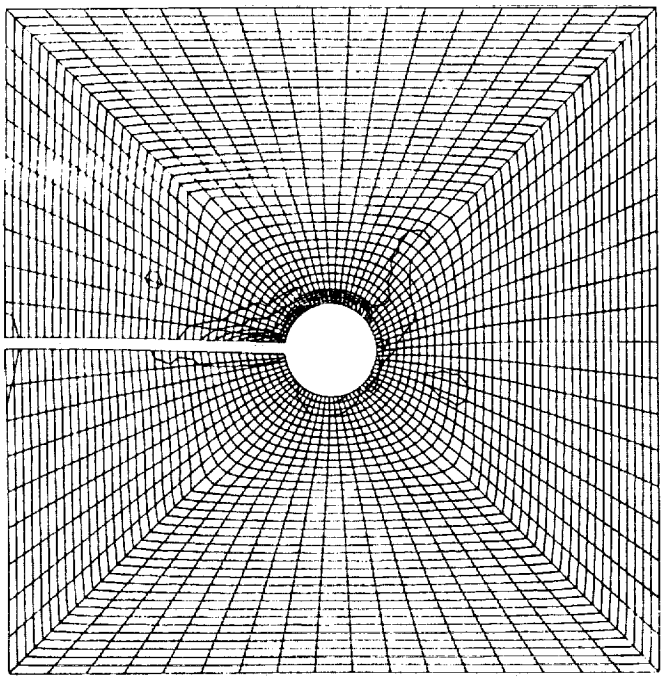
Figure 5.34c: Evolution of density contours for Case 2 parameter study: Orthogonality = 0.40, Smoothness = 0.60. Numbers under figures refer to time step.



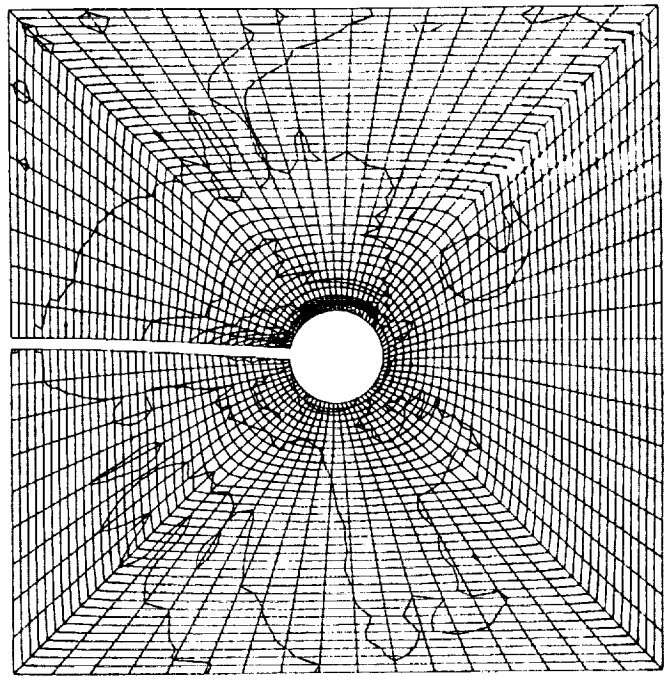
650



700

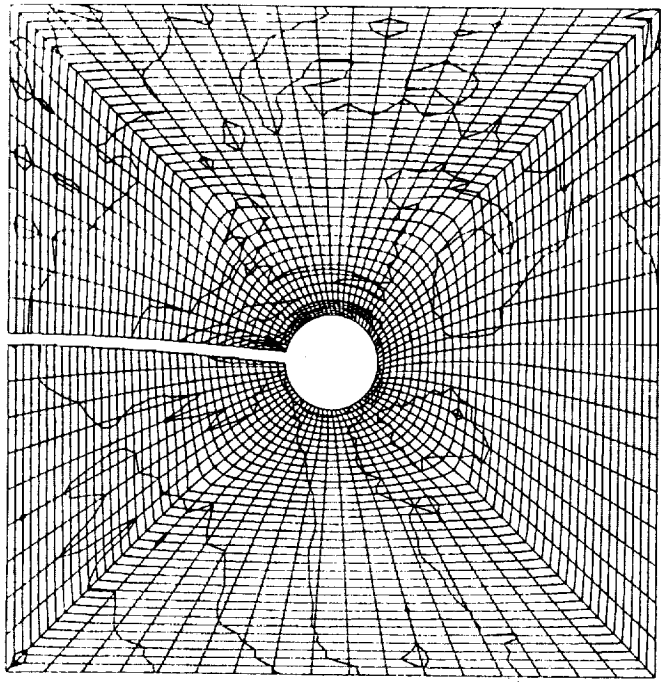


750

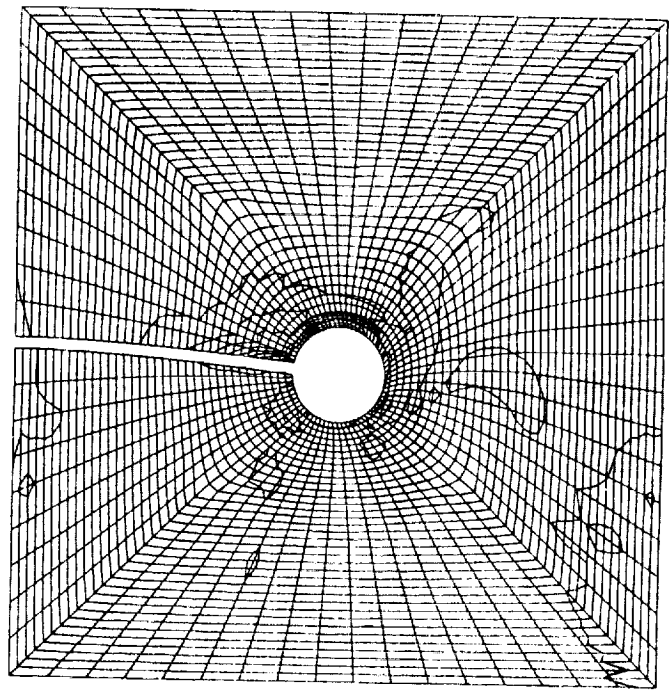


800

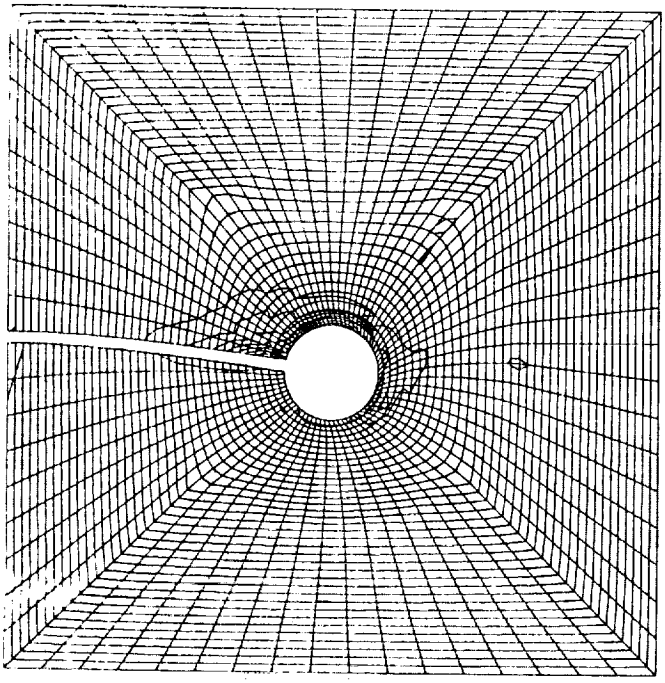
Figure 5.34d: Evolution of density contours for Case 2 parameter study: Orthogonality = 0.40, Smoothness = 0.60. Numbers under figures refer to time step.



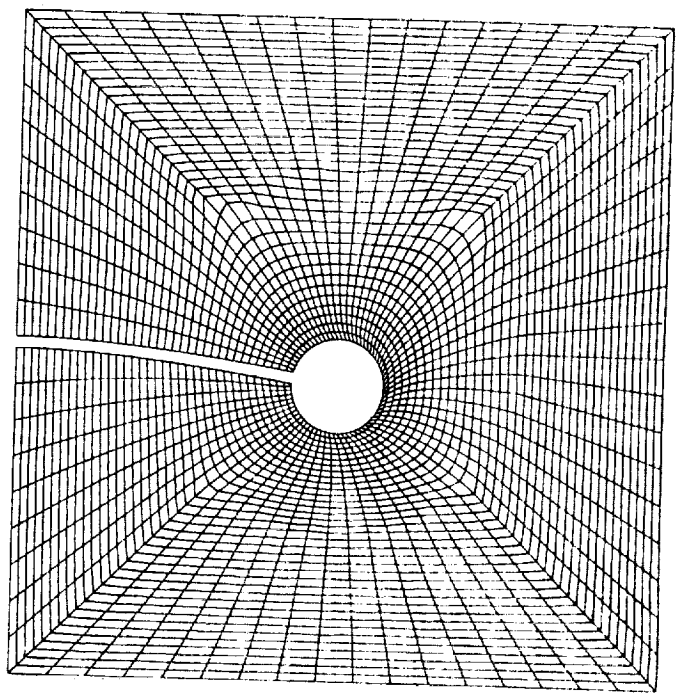
850



900



950



1000

Figure 5.34e: Evolution of density contours for Case 2 parameter study: Orthogonality = 0.40, Smoothness = 0.60. Numbers under figures refer to time step.

6 SUMMARY AND CONCLUSIONS

Two adaptive methods have been implemented to investigate a pair of complex fluid-structure interaction problems. The first method, the quasi-steady-state method, appears to be quite satisfactory for determining steady–state conditions. However, this method at present requires a great deal of user interaction and is computational inefficient. With this approach, the updated grid is completely regenerated for steady–state conditions without regard to any previous solutions. These drawbacks, however, may be partially overcome by automating the regridding procedure and interpolating nodal point solution values from previous grids. Even with these enhancements this method appears to be less computationally efficient than the node relocation method and will also be unable to provide time accurate transient solutions unless regridding occurs at every step.

The node relocation method at the present time appears to be quite promising. It is fully automated, requiring no interaction by the user, and is capable of producing transient results for some very complex configurations. Additional work, however, is needed to implement the final steps of the algorithm as outlined in Section 4.2. When completed, this method should provide an extremely versatile approach for solving a large class of transient fluid–structure interaction problems.

7 REFERENCES

1. Donea, J., Giuliani, S., and Halleux, J. P., "An Arbitrary Lagrangian-Eulerian Finite Element Method for Transient Dynamic Fluid-Structure Interactions," Computer Methods in Applied Mechanics and Engineering, (1982), 689–723.

2. Steger, Joseph L., Dougherty, F. Carroll, and Benek, John A., "A Chimera Grid Scheme," ASME Mini–Symposium on Advances in Grid Generation, Houston, Texas, June 1983.

3. Benek, J. A., Steger, J. L., and Dougherty, F. C., "A Flexible Grid Embedding Technique With Application to the Euler Equations," Computational Fluid Dynamics Conference, 6th, Danvers, Massachusetts, July 13–15, 1983.

4. Atluri, Satya N. and Nishioka, T., "Numerical Studies in Dynamic Fracture Mechanics," Int. J. Frac., **27**, (1985) 245–261.

5. Lynch, Daniel R., "Unified Approach to Simulation on Deforming Elements with Application to Phase Change Problems," J. Comp. Phys., **47**, (1982) 387–411.

6. Jacquotte, Olivier P., "A Mechanical Model for a New Grid Generation Method in Computational Fluid Dynamics," Comp. Meth. Appl. Mech Eng., **66**, (1988) 323–338.

7. Kukuchi, Noboru, and Chang, Jung-Ho, "Adaptive Remodeling of Finite Element Grids for Large Deformation Elastic–Plasticity in Metal Forming Analysis."

8. Thompson, J. F., Thames, F. C., and Mastin, C. M., "Automatic Numerical Generation of Body–Fitted Curvilinear Coordinate System for Field Containing Any Number of Arbitrary Two–Dimensional Bodies," J. of Comp. Physics, Vol. 15, pp. 299–319, 1974.

9. Steger, J. S. and Chaussee, D. S., "Generation of Body Fitted Coordinates Using Hyperbolic Partial Differential Equations," FSI Report 80–1, Flow Simulation, Inc., Sunnyvale, California, January, 1980.

10. S. Nakamura, "Marching Grid Generation Using Parabolic Partial Differential Equations," NCA2–OR565–101 Report, 1982.

11. Carcaillet, Richard, Dulikravich, George S., and Kennon, Stephen R., "Generation of Solution–Adaptive Computational Grids Using Optimization," Comp. Meth. Appl. Mech. Eng., **57** (1986), 279–295.

12. Jameson, A., Schmidt, W., and Turkel, E., "Numerical Solutions of the Euler Equations by Finite Volume Methods Using Runge–Kutta Time–Stepping Schemes," AIAA Paper, 81–1259, Palo Alto, California, June 23–25, 1981.

13. Lo, S. H., "A New Mesh Generation Scheme for Arbitrary Planar Domains," Intl. J. Num. Meth. Eng., **21** (1985), 1403–1426.

A APPENDIX: LISTING OF THE REMESHING ALGORITHM

This appendix contains an alphabetical listing of the major subroutines constructed for the execution of the remeshing algorithm. The initial call to the remeshing package is the call to subroutine CALCMESH in the pre–processing area of the code. This routine initializes the remeshing tolerances. After the implementation of the boundary displacements affected by the call to the structures package, the remeshing algorithm is accessed with a call to the driver routine REMESH. Subroutine CHKMESH tests Jacobians, side lengths, and diagonals against the tolerances generated in CALCMESH. Subroutines LOADCOF1 and LOADCOF2 load up the various coefficients needed for solution of the ψ equation (see Eqs.4.2 and 4.3) which is calculated in subroutine REALROOT. When all new node locations have been generated, the subroutine INTERPOL is called to drive the interpolation process. Routine ISPTHERE determines the element that contains the new location of the central master element node. Values from this element are used to interpolate values at the new node location in subroutine INTHERE and INTQTN.

The following is a table of contents for the remainder of this appendix.

```
      SUBROUTINE CALCMESH
C
C.....................................................................
C                                                                     .
C THIS ROUTINE CALCULATES A TOLERANCE VALUE FOR THE REMESHING         .
C CRITERION.                                                          .
C                                                                     .
C.....................................................................
C
%INCLUDE 'SYSCOM.BLK'
%INCLUDE 'PARAMS.BLK'
%INCLUDE 'CCON.BLK'
%INCLUDE 'MOVE.BLK'
%INCLUDE 'CELEM.BLK'
%INCLUDE 'CNODE.BLK'
%INCLUDE 'INFLUNC.BLK'
%INCLUDE 'MESHCOEF.BLK'
C
      DIMENSION XCORD(2,4)
C
C
C SET INITIAL TOLERANCE LEVEL TO A LARGE NUMBER
C
      TOLMESH=1000000
      TOLMESH1=1000000
C
C SET THE FRACTION
C
      FRAC=0.90
C
C LOOP OVER THE ELEMENTS
C
      DO 1000 IEL=1,NELEM
C
C TEST TO SEE IF THE ELEMENT IS IN THE INFLUENCE REGION
C
         IF(IELACT(IEL).GT.1)THEN
C
C DETERMINE THE COORDINATES OF THE NODES AND THE DIAGONALS OF THE
C ELEMENTS
            DO 100 INOD=1,4
               NODE=NODES(INOD,IEL)
               XCORD(1,INOD)=X(1,NODE)
               XCORD(2,INOD)=X(2,NODE)
100         CONTINUE
C
            DIAG1=SQRT((XCORD(1,3)-XCORD(1,1))**2
     .           +(XCORD(2,3)-XCORD(2,1))**2)
            DIAG2=SQRT((XCORD(1,4)-XCORD(1,2))**2
     .           +(XCORD(2,4)-XCORD(2,2))**2)
C
C FIND THE RATIO OF THE SHORTER TO LONGER SIDE
C
            SHORT=MIN(DIAG1,DIAG2)
            XLONG=MAX(DIAG1,DIAG2)
            TEST=SHORT/XLONG
C
C FIND THE MINIMUM
C
            TOLMESH=MIN(TOLMESH,TEST)
C
C CALCULATE THE SIDE LENGTHS
```

```
C
            SIDE1=SQRT((XCORD(1,2)-XCORD(1,1))**2
     .                 +(XCORD(2,2)-XCORD(2,1))**2)
            SIDE2=SQRT((XCORD(1,3)-XCORD(1,2))**2
     .                 +(XCORD(2,3)-XCORD(2,2))**2)
            SIDE3=SQRT((XCORD(1,4)-XCORD(1,3))**2
     .                 +(XCORD(2,4)-XCORD(2,3))**2)
            SIDE4=SQRT((XCORD(1,1)-XCORD(1,4))**2
     .                 +(XCORD(2,1)-XCORD(2,4))**2)
C
C FIND THE SHORTEST AND LONGEST SIDES
C
            SMAX=MAX(SIDE1,SIDE2,SIDE3,SIDE4)
            SMIN=MIN(SIDE1,SIDE2,SIDE3,SIDE4)
C
C FIND THE RATIO OF THE SHORTER TO LONGER SIDE
C
            TESTSID=SMIN/SMAX
C
C SAVE THE MINIMUM VALUE
C
            TESTMIN1=MIN(TESTSID,TESTMIN1)
C
         ENDIF
C
1000  CONTINUE
C
C SET THE TOLERANCE AS A FRACTION OF THE MINIMUM TOLERANCE OF THE
C ORIGINAL MESH
C
      TOLMESH=TOLMESH*FRAC
      TOLMESH1=TOLMESH1*FRAC
C
C
      RETURN
      END


      SUBROUTINE CHKMESH(IFLAG)
C
C.....................................................................
C                                                                     .
C THIS ROUTINE DETERMINES IF THE INFLUENCE MESH NEEDS                 .
C REMESHING.  REMESHING IS PERFORMED IF THE RATIO OF THE              .
C DIAGONALS OF AT LEAST ONE ELEMENT IN THE INFLUENCE                  .
C REGION IS LESS THAN A STATED TOLERANCE VALUE.                       .
C                                                                     .
C.....................................................................
C
C PARAMETERS:
C     IFLAG - FLAG TO DETERMINE IF REMESHING IS TO BE PERFORMED
C             1,2 = YES, 0 = NO
C
%INCLUDE 'SYSCOM.BLK'
%INCLUDE 'PARAMS.BLK'
%INCLUDE 'LUNITS.BLK'
%INCLUDE 'CCON.BLK'
%INCLUDE 'MOVE.BLK'
%INCLUDE 'CELEM.BLK'
%INCLUDE 'CNODE.BLK'
%INCLUDE 'INFLUNC.BLK'
```

```
%INCLUDE 'MESHCOEF.BLK'
%INCLUDE 'QUADR.BLK'
%INCLUDE 'TRANSFO.BLK'
C
      DIMENSION XC(4),YC(4)
      DIMENSION NODCON(4),NODSEC(4),XIP(2,4)
C
      DATA XIP/-1.0,-1.0,  1.0,-1.0,  1.0,1.0,  -1.0,1.0/
C
C
C INITIALIZE THE IFLAG TO ZERO
C
      IFLAG=0
      TESTMIN=1.0E6
      TESTMIN1=1.0E6
      SMALL  =1.0E-5
      XJMIN=1.0E6
C
C RETURN IF THERE ARE NO ELEMENTS IN THE INFLUENCE REGION
C
      IF(NNINFR.EQ.0)RETURN
C
C LOOP OVER THE ELEMENTS AND DETERMINE WHICH ARE IN THE
C INFLUENCE REGION
C
      DO 1000 IEL=1,NELEM
C
         IF(IELACT(IEL).GT.1)THEN
C
C DETERMINE THE ELEMENT AND THE LENGTHS OF THE DIAGONALS
C ======================================================
C
            CALL DETCO(IEL,NUMCON,NODCON,NODSEC)
            CALL COORDTN1(IEL,NUMCON,NODCON,NODSEC,XC,YC)
C
            DIAG1=SQRT((XC(3)-XC(1))**2+(YC(3)-YC(1))**2)
            DIAG2=SQRT((XC(4)-XC(2))**2+(YC(4)-YC(2))**2)
C
C FIND THE RATIO OF THE SHORTER TO LONGER DIAGONAL
C
            SHORT=MIN(DIAG1,DIAG2)
            XLONG=MAX(DIAG1,DIAG2)
            TEST=SHORT/XLONG
C
C SAVE THE MINIMUM VALUE
C
            TESTMIN=MIN(TEST,TESTMIN)
C
C DETERMINE THE LENGTHS OF THE SIDES
C ==================================
C
            SIDE1=SQRT((XC(2)-XC(1))**2+(YC(2)-YC(1))**2)
            SIDE2=SQRT((XC(3)-XC(2))**2+(YC(3)-YC(2))**2)
            SIDE3=SQRT((XC(4)-XC(3))**2+(YC(4)-YC(3))**2)
            SIDE4=SQRT((XC(1)-XC(4))**2+(YC(1)-YC(4))**2)
C
C FIND THE SHORTEST AND LONGEST SIDES
C
            SMAX=MAX(SIDE1,SIDE2,SIDE3,SIDE4)
            SMIN=MIN(SIDE1,SIDE2,SIDE3,SIDE4)
C
C FIND THE RATIO OF THE SHORTER TO LONGER SIDE
```

```
C
            TESTSID=SMIN/SMAX
C
C SAVE THE MINIMUM VALUE
C
            TESTMIN1=MIN(TESTSID,TESTMIN1)
C
C CHECK THE JACOBIAN AT THE INTEGRATION POINTS
C ============================================
C
            CALL TRANSF(XC,YC)
C
C LOOP OVER THE INTEGRATION POINTS
C
            DO 100 INT=1,4
C
C CALCULATE THE DERVATIVES OF X AND Y WRT THE MASTER COORDS
C
               DXDS11=CXXO+CXXHO*XIP(2,INT)
               DXDS12=CXHO+CXXHO*XIP(1,INT)
               DXDS21=CYXO+CYXHO*XIP(2,INT)
               DXDS22=CYHO+CYXHO*XIP(1,INT)
C
C COMPUTE THE JACOBIAN OF THE TRANSFORMATION.
C
               XJAC=DXDS11*DXDS22-DXDS12*DXDS21
C
C WRITE OUT THE BAD VALUES
C
               IF(XJAC.LT.0.0)THEN
                  WRITE(LTERM,1100)IEL,INT
1100              FORMAT(/,/,3X,'CHKELM : BAD JACOBIAN ',
                        'AT ELEMENT, INT ',2I7)
               ENDIF
C
C TEST FOR MINIMUM JACOBIAN
C
               XJMIN=MIN(XJAC,XJMIN)
C
100         CONTINUE
C
         ENDIF
C
1000  CONTINUE
C
C
C
C SET A FLAG FOR MOVING THE NODES
C ===============================
C
C CHECK THE MINIMUM VALUES FOR A VALUE LESS THAN THE TOLERANCE
C
      IF(TESTMIN.LT.TOLMESH)IFLAG=1
      IF(TESTMIN1.LT.TOLMESH1)IFLAG=1
C
      IF(XJMIN.LT.SMALL)IFLAG=2
C
C
      RETURN
      END
```

```
      SUBROUTINE INTERPOL
%INCLUDE 'SYSCOM.BLK'
C
C.....................................................................
C                                                                     .
C THIS ROUTINE INTERPOLATES THE VALUES AT THE NEW LOCATIONS OF THE NODES .
C FROM THE VALUES AT THE OLD LOCATIONS OF THE NODES.                  .
C                                                                     .
C.....................................................................
C
%INCLUDE 'PARAMS.BLK'
%INCLUDE 'BUFFER.BLK'
%INCLUDE 'CELEM.BLK'
%INCLUDE 'INFLUNC.BLK'
%INCLUDE 'LUNITS.BLK'
C
      DIMENSION V(2)
      DIMENSION LIST(2,1000)
C
C LOOP THROUGH INTERIOR BOUNDARY NODES
C
      DO 1000 INODE = 1, NNINFR
C
C INITIALIZE FLAGS AND ARRAYS
C
         IFLAG  = 0
         IHERE  = 0
         ICOUNT = 1
         CALL SETZI(2000, LIST)
C
C GET ELEMENT, NODE NUMBERS AND NODAL COORDINATES
C
         MNODE = INTNOD(1,INODE)
         IEL   = INTNOD(2,INODE)
         V(1)  = XBUF(1,MNODE)
         V(2)  = XBUF(2,MNODE)
C
C CHECK TO SEE IF THE NEW LOCATION OF THIS NODE FALLS INSIDE IEL
C
         CALL ISPTHERE(IEL, V, IHERE)
C
C IF MNODE IS IN THIS ELEMENT, INTERPOLATE THE VALUES AND RESET QTN
C
         IF (IHERE .EQ. 1) THEN
C
            CALL INTHERE(IEL, MNODE, V)
            GOTO 1000
C
         ELSE
C
C IF MNODE IS NOT IN THE ELEMENT, ADD TO LIST AND SET CHECK FLAG
C
            LIST(1,ICOUNT) = IEL
            LIST(2,ICOUNT) = 1
C
         ENDIF
C
C ALLOW SIX LEVELS OF ITERATIONS
C
         DO 800 ILEV = 1, 6
C
```

```
C SAVE THE CURRENT NUMBER OF ELEMENTS IN THE LIST
C
            ICOUNT2  = ICOUNT
C
C LOOP OVER THE ELEMENTS IN THE LIST
C
            DO 250 I = 1, ICOUNT2
C
C RETRIEVE THE ELEMENT NUMBER
C
               IEL = LIST(1,I)
C
C LOOP OVER THE NELCON SIDES OF THE ELEMENT
C
               DO 200 J = 1, 8
C
C FIND A NEIGHBORING ELEMENT
C
                  NEIGH = NELCON(J,IEL)
C
                  IF (NEIGH .GT. 0) THEN
C
C CHECK TO SEE IF THE ELEMENT IS IN THE PREVIOUS LIST
C
                     CALL SEARCHI(LIST, 2, 1, ICOUNT2, NEIGH, INLIST)
C
C IF IT'S NOT IN THE LIST, ADD IT TO THE LIST
C
                     IF (INLIST .EQ. 0) THEN
C
                        ICOUNT = ICOUNT + 1
                        LIST(1,ICOUNT) = NEIGH
                        LIST(2,ICOUNT) = 0
C
                     ENDIF
C
                  ENDIF
C
200            CONTINUE
C
250         CONTINUE
C
C SORT THE LIST AND ELIMINATE DUPLICATES
C
            CALL SORTI(LIST, 2, 1, ICOUNT)
            CALL ELMDUP(LIST, 2, 1, ICOUNT)
C
C LOOP OVER THE ELEMENTS IN THE LIST
C
            DO 300 I = 1, ICOUNT
C
C CHECK TO SEE IF THIS ELEMENT HAS BEEN CHECKED
C
               IF (LIST(2,I) .EQ. 0) THEN
C
C RETRIEVE THE ELEMENT NUMBER
C
                  IEL = LIST(1,I)
C
C CHECK TO SEE IF THE NEW LOCATION OF THIS NODE FALLS INSIDE IEL
C
                  CALL ISPTHERE(IEL, V, IHERE)
```

```
C
C IF MNODE IS IN THIS ELEMENT, INTERPOLATE THE VALUES AND RESET QTN
C
                    IF (IHERE .EQ. 1) THEN
C
                       CALL INTHERE(IEL, MNODE, V)
                       GOTO 1000
C
                    ELSE
C
C IF MNODE IS NOT IN THE ELEMENT SET CHECK FLAG
C
                       LIST(2,I) = 1
C
                    ENDIF
C
                ENDIF
C
300           CONTINUE
C
800       CONTINUE
C
C IF THE 800 LOOP HAS BEEN COMPLETED WITHOUT JUMPING OUT, THE NODE
C HAS NOT BEEN FOUND IN THE AREA, SO WRITE AN ERROR MESSAGE AND STOP
C
          WRITE(LTERM,900)V(1),V(2),MNODE
900        FORMAT(/,/,3X,' POINT NOT FOUND ',2E15.7,'     MNODE = ',I5)
          CALL SSTOP(0,'INTERPOL')
C
1000   CONTINUE
C
C
       RETURN
       END



       SUBROUTINE INTHERE(IEL, MNODE, V)
%INCLUDE 'SYSCOM.BLK'
C
C.....................................................................
C                                                                     .
C THIS ROUTINE INTERPOLATES THE VALUES AT A NODE IN AN ELEMENT AND RESETS .
C THE CORRESPONDING QTN RECORD.                                       .
C                                                                     .
C.....................................................................
C
%INCLUDE 'PARAMS.BLK'
%INCLUDE 'SOLVEC.BLK'
C
       DIMENSION QTNVAL(4)
C
C CALL THE INTERPOLATION ROUTINE
C
       CALL INTQTN(IEL, V, QTNVAL)
C
C RESET THE QTN VALUES
C
       DO 100 NDOF = 1, 4
C
          QTN(MNODE,NDOF) = QTNVAL(NDOF)
C
```

```
100   CONTINUE
C
C
      RETURN
      END


      SUBROUTINE INTQTN(IEL,XY,QTNLOC)
C
C.....................................................
C                                                     .
C THIS ROUTINE EXTRACTS THE SOLUTION VALUE            .
C AT THE POINT XY                                     .
C                                                     .
C.....................................................
C
C PARAMETERS:
C     XY      - COORDINATES OF THE POINT
C     QTNLOC - EXTRACTED SOLUTION VALUE
C
%INCLUDE 'SYSCOM.BLK'
%INCLUDE 'PARAMS.BLK'
%INCLUDE 'LUNITS.BLK'
C
      DIMENSION XY(2),QTNLOC(4),EPS(2)
      DIMENSION QNOD(4,4),PSI(4),DPSI(2,4)
C
C LOAD UP THE MASTER ELEMENT COORDINATE POINT
C
      CALL GETEN(IEL,XY(1),XY(2),EPS)
C
C LOAD UP THE SOLUTION VALUES
C
      CALL GETSOL(IEL,QNOD)
C
C EVALUATE THE PRIMATIVE VARIABLES FROM SUMMING THE
C SHAPE FUNCITONS TIMES THE NODAL VALUES
C
      CALL SETZ(4,QTNLOC)
      CALL SHAPE(EPS,PSI,DPSI)
C
      DO 500 IFLOW=1,4
         DO 400 ICOMP=1,4
            QTNLOC(IFLOW)=QTNLOC(IFLOW)+QNOD(IFLOW,ICOMP)*PSI(ICOMP)
400      CONTINUE
500   CONTINUE
C
C
      RETURN
      END


      SUBROUTINE ISPTHERE(IEL,V,IFLAG)
C
C.....................................................................
C                                                                     .
C THE ROUTINE DETERMINES IF THE GIVEN LOCATION FALLS WITHIN           .
C THE SPECIFIED NODE  (IS PoinT HERE)                                 .
C                                                                     .
C.....................................................................
```

```
C
C PARAMETERS:
C      IEL    - SPECIFIED ELEMENT
C      V      - COORDINATES OF POINT FOR SEARCH
C      IFLAG - FLAG SET TO 1 IF LOCATION IS IN ELEMENT
C
%INCLUDE 'SYSCOM.BLK'
%INCLUDE 'PARAMS.BLK'
C
      DIMENSION XC(4),YC(4),VEC(2,4),V(2)
      DIMENSION NUMCON(4),NODSEC(4)
C
C INITIALIZE THE FLAG
C
      IFLAG=0
C
C DETERMINE THE COORDINATES OF THE ELEMENT
C
      CALL DETCO(IEL,NUMCON,NODCON,NODSEC)
      CALL COORD(IEL,NUMCON,NODCON,NODSEC,XC,YC)
C
C DETERMINE THE COMPONENTS OF THE VECTORS FROM THE
C COORDINATE POINT TO THE NODES
C
      DO 100 ICO=1,4
         VEC(1,ICO)=XC(ICO)-V(1)
         VEC(2,ICO)=YC(ICO)-V(2)
100   CONTINUE
C
C CHECK AND SEE IF THE COORD POINT IS IN THE FIRST
C TRIANGULAR SUBELEMENT DEFINED BY POINTS 1,2, AND 4
C
C CALCULATE THE NECESSARY CROSS PRODUCTS
C V1 X V2     V2 X V4    V4 X V1
C
      CROSS12=VEC(1,1)*VEC(2,2)-VEC(1,2)*VEC(2,1)
      CROSS24=VEC(1,2)*VEC(2,4)-VEC(1,4)*VEC(2,2)
      CROSS41=VEC(1,4)*VEC(2,1)-VEC(1,1)*VEC(2,4)
C
C CHECK AND SEE IF ALL THE CROSS PRODUCTS ARE >= 0
C
      IF(CROSS12.GE.0.AND.CROSS24.GE.0.AND.CROSS41.GE.0)THEN
         IFLAG=1
         RETURN
      ENDIF
C
C THE POINT IS NOT IN THE FIRST SUBELEMENT CHECK THE SECOND
C TRIANGULAR ELEMENT DEFINED BY POINTS 2,3, AND 4
C
C CALCULATE THE NECESSARY CROSS PRODUCTS
C V2 X V3      V3 X V4      V4 X V2
C
      CROSS23=VEC(1,2)*VEC(2,3)-VEC(1,3)*VEC(2,2)
      CROSS34=VEC(1,3)*VEC(2,4)-VEC(1,4)*VEC(2,3)
      CROSS42=VEC(1,4)*VEC(2,2)-VEC(1,2)*VEC(2,4)
C
C CHECK AND SEE IF ALL THE CROSS PRODUCTS ARE >= 0
C
      IF(CROSS23.GE.0.AND.CROSS34.GE.0.AND.CROSS42.GE.0)THEN
         IFLAG=1
         RETURN
      ENDIF
```

```
C
C
      RETURN
      END


      SUBROUTINE LOADCOF1(A)
C
C.....................................................................
C                                                                     .
C THIS ROUTINE LOADS UP THE VARIOUS COEFFICIENTS USED IN THE          .
C REMESHING EQUATIONS.  THE COEFFICIENTS K, L, AND M ARE FROM         .
C THE SMOOTHNESS EQUATION.  THE COEFFICIENTS P AND R ARE FROM         .
C THE ORTHOGONALITY EQUATION.  THE C COEFFICIENTS ARE USED IN         .
C THE F FUNCTION EQUATION AND ITS GRADIENT.                           .
C                                                                     .
C.....................................................................
C
C PARAMETERS:
C     A - COORDINATES OF THE NODES NEIGHBORING THE MASTER NODE
C
%INCLUDE 'SYSCOM.BLK'
%INCLUDE 'PARAMS.BLK'
%INCLUDE 'MESHCOEF.BLK'
C
      DIMENSION A(2,4)
      DIMENSION XK(4,3),XL(12),XM(6)
      DIMENSION P(12),R(8)
C
C LOAD UP THE K COEFFICIENTS
C
      DO 100 I=1,4
         J=I+1
         IF(J.GT.4)J=J-4
         XK(I,1)=A(1,I)*A(2,J)-A(2,I)*A(1,J)
         XK(I,2)=A(2,I)-A(2,J)
         XK(I,3)=A(1,J)-A(1,I)
100   CONTINUE
C
C LOAD UP THE L COEFFICIENTS
C
      DO 200 IDUM=1,4
         IFACTOR=IDUM-1
         IDUM1=IDUM+1
         IF(IDUM1.GT.4)IDUM1=IDUM1-4
         DO 250 I=1,3
            ISUB=I+3*IFACTOR
            XL(ISUB)=XK(IDUM,I)-XK(IDUM1,I)
250      CONTINUE
200   CONTINUE
C
C LOAD UP THE M COEFFICIENTS
C
      XM(1)=XL(1)*XL(1)+XL(4)*XL(4)+XL(7)*XL(7)+XL(10)*XL(10)
      XM(2)=2.0*(XL(1)*XL(2)+XL(4)*XL(5)+XL(7)*XL(8)+XL(10)*XL(11))
      XM(3)=2.0*(XL(1)*XL(3)+XL(4)*XL(6)+XL(7)*XL(9)+XL(10)*XL(12))
      XM(4)=XL(2)*XL(2)+XL(5)*XL(5)+XL(8)*XL(8)+XL(11)*XL(11)
      XM(5)=XL(3)*XL(3)+XL(6)*XL(6)+XL(9)*XL(9)+XL(12)*XL(12)
      XM(6)=2.0*(XL(2)*XL(3)+XL(5)*XL(6)+XL(8)*XL(9)+XL(11)*XL(12))
C
C LOAD UP THE P COEFFICIENTS
```

```
C
      P(1)=A(1,1)*A(1,2)+A(2,1)*A(2,2)
      P(2)=-A(1,1)-A(1,2)
      P(3)=-A(2,1)-A(2,2)
      P(4)=A(1,4)*A(1,1)+A(2,4)*A(2,1)
      P(5)=-A(1,4)-A(1,1)
      P(6)=-A(2,4)-A(2,1)
      P(7)=A(1,3)*A(1,4)+A(2,3)*A(2,4)
      P(8)=-A(1,3)-A(1,4)
      P(9)=-A(2,3)-A(2,4)
      P(10)=A(1,2)*A(1,3)+A(2,2)*A(2,3)
      P(11)=-A(1,2)-A(1,3)
      P(12)=-A(2,2)-A(2,3)
C
C LOAD UP THE R COEFFICIENTS
C
      R(1)=P(1)*P(1)+P(4)*P(4)+P(7)*P(7)+P(10)*P(10)
      R(2)=2.0*(P(1)*P(2)+P(4)*P(5)+P(7)*P(8)+P(10)*P(11))
      R(3)=2.0*(P(1)*P(3)+P(4)*P(6)+P(7)*P(9)+P(10)*P(12))
      R(4)=2.0*(P(1)+P(4)+P(7)+P(10))+P(2)*P(2)+P(5)*P(5)
     .        +P(8)*P(8)+P(11)*P(11)
      R(5)=2.0*(P(1)+P(4)+P(7)+P(10))+P(3)*P(3)+P(6)*P(6)
     .        +P(9)*P(9)+P(12)*P(12)
      R(6)=2.0*(P(2)*P(3)+P(5)*P(6)+P(8)*P(9)+P(11)*P(12))
      R(7)=2.0*(P(2)+P(5)+P(8)+P(11))
      R(8)=2.0*(P(3)+P(6)+P(9)+P(12))
C
C LOAD THE C COEFFICIENTS INTO MESHCOEF.BLK
C
      DO 300 I=1,6
         C(I)=XM(I)+ALPHA*(R(I)-XM(I))
300   CONTINUE
C
      C(7)=ALPHA*R(7)
      C(8)=ALPHA*R(8)
      C(9)=4.0*ALPHA
      C(10)=2*C(9)
C
C
      RETURN
      END



      SUBROUTINE LOADCOF2(V,DV)
C
C.....................................................................
C                                                                     .
C THIS ROUTINE CALCULATES THE COEFFICIENTS FOR THE PSI                .
C EQUATION AND ITS DERIVATIVE USED IN THE REMESHING ROUTINES          .
C                                                                     .
C.....................................................................
C
C PARAMETERS:
C     V  - COORDINATES OF THE MASTER NODE
C     DV - CHANGE IN COORDINATES OF THE MASTER NODE
C
%INCLUDE 'SYSCOM.BLK'
%INCLUDE 'PARAMS.BLK'
%INCLUDE 'MESHCOEF.BLK'
C
      DIMENSION V(2),DV(2)
```

```
C
      G(1)=C(1)+C(2)*V(1)+C(3)*V(2)+C(4)*V(1)*V(1)+C(5)*V(2)*V(2)
     .    +C(6)*V(1)*V(2)+C(7)*V(1)*V(1)*V(1)+C(8)*V(1)*V(1)*V(2)
     .    +C(7)*V(1)*V(2)*V(2)+C(8)*V(2)*V(2)*V(2)
     .    +C(9)*(V(1)*V(1)*V(1)*V(1)+V(2)*V(2)*V(2)*V(2))
     .    +C(1)*V(1)*V(1)*V(2)*V(2)
C
      G(2)=C(2)*DV(1)+C(3)*DV(2)+2.0*C(4)*V(1)*DV(1)
     .    +2.0*C(5)*V(2)*DV(2)+C(6)*(V(2)*DV(1)+V(1)*DV(2))
     .    +3.0*C(7)*V(1)*V(1)*DV(1)+C(8)*(2.0*V(1)*V(2)*DV(1)
     .    +V(1)*V(1)*DV(2))+C(7)*(2.0*V(1)*V(2)*DV(2)
     .    +V(2)*V(2)*DV(1))+3.0*C(8)*V(2)*V(2)*DV(2)
     .    +C(9)*4.0*(V(1)*V(1)*V(1)*DV(1)+V(2)*V(2)*V(2)*DV(2))
     .    +C(10)*2.0*(V(1)*V(1)*V(2)*DV(2)+V(1)*V(2)*V(2)*DV(1))
C
      G(3)=C(4)*DV(1)*DV(1)+C(5)*DV(2)*DV(2)+C(6)*DV(1)*DV(2)
     .    +3.0*C(7)*V(1)*DV(1)*DV(1)+C(8)*(V(2)*DV(1)*DV(1)
     .    +2.0*V(1)*DV(1)*DV(2))+C(7)*(V(1)*DV(2)*DV(2)
     .    +2.0*V(2)*DV(1)*DV(2))+3.0*C(8)*V(2)*DV(2)*DV(2)
     .    +C(9)*6.0*(V(1)*V(1)*DV(1)*DV(1)+V(2)*V(2)*DV(2)*DV(2))
     .    +C(10)*(V(1)*V(1)*DV(2)*DV(2)+V(2)*V(2)*DV(1)*DV(1)
     .    +4.0*V(1)*V(2)*DV(1)*DV(2))
C
      G(4)=C(7)*DV(1)*DV(1)*DV(1)+C(8)*DV(1)*DV(1)*DV(2)
     .    +C(7)*DV(1)*DV(2)*DV(2)+C(8)*DV(2)*DV(2)*DV(2)
     .    +C(9)*4.0*(V(1)*DV(1)*DV(1)*DV(1)+V(2)*DV(2)*DV(2)*DV(2))
     .    +C(10)*2.0*(V(1)*DV(1)*DV(2)*DV(2)+V(2)*DV(1)*DV(1)*DV(2))
C
      G(5)=C(9)*(DV(1)*DV(1)*DV(1)*DV(1)+DV(2)*DV(2)*DV(2)*DV(2))
     .    +C(10)*DV(1)*DV(1)*DV(2)*DV(2)
C
C
      RETURN
      END



      SUBROUTINE REALROOT(COEF,ROOTS,ISAME,ROOT)
C
C.....................................................
C                                                     .
C THIS ROUTINE SORTS THROUGH THE ARRAY ROOTS          .
C AND DETERMINES THE NUMBER OF REAL ROOTS AND         .
C PLACES THE RESULTS INTO RROOTS                      .
C                                                     .
C.....................................................
C
C PARAMETERS:
C     COEF      - COEFFICIENTS FOR THE FOURTH ORDER POLYNOMIAL
C     ROOTS     - ROOTS FROM ZCUBIC REAL AND IMAGINARY PARTS
C     ISAME     - FLAG INDICATING THE NUMBER OF IDENTICAL ROOTS
C     ROOT      - REAL ROOTS
C
%INCLUDE 'SYSCOM.BLK'
%INCLUDE 'PARAMS.BLK'
%INCLUDE 'LUNITS.BLK'
C
      DIMENSION COEF(5),ROOTS(2,3),RROOTS(3)
C
C
C DEFINE A SMALL VALUE
C
```

```
      SMALL=1.0E-7
      BIG  =1.0E6
      OTEN =0.1
      ONE  =1.0
C
C CHECK AND SEE IF ISAME IS GREATER THAN 1
C IF SO THEN ALL THE ROOTS MUST BE REAL
C
      IF(ISAME.GE.2)THEN
C
C SET THE NUMBER OF REAL ROOTS AND THE VALUES
C
         NREAL=3
         RROOTS(1)=ROOTS(1,1)
         RROOTS(2)=ROOTS(1,2)
         RROOTS(3)=ROOTS(1,3)
         GOTO100
C
      ELSE
C
C THERE ARE NO EQUAL ROOTS LOOK AN IMAGINARY ROOT
C
         RMAX=MAX(ROOTS(2,1),ROOTS(2,2),ROOTS(2,3))
C
C CHECK AND SEE IF THE MAX ROOT IS LESS THAN SMALL
C
         IF(RMAX.LT.SMALL)THEN
C
C ASSUME ALL THREE ROOTS ARE REAL
C
            NREAL=3
            RROOTS(1)=ROOTS(1,1)
            RROOTS(2)=ROOTS(1,2)
            RROOTS(3)=ROOTS(1,3)
            GOTO100
C
         ELSE
C
C CALCULATE THE RELATIVE NORMS OF THE IMAGINARY PARTS
C
            XNORM1=ROOTS(2,1)/RMAX
            XNORM2=ROOTS(2,2)/RMAX
            XNORM3=ROOTS(2,3)/RMAX
C
C CHECK FOR EQUAL VALUES AND OPPOSITE SIGN OF 1 AND 2
C
            IF(ABS(ABS(XNORM1)-ABS(XNORM2)).LT.OTEN)THEN
C
C CHECK FOR OPPOSITE SIGNS
C
               SIGN1=SIGN(ONE,XNORM1)
               SIGN2=SIGN(ONE,XNORM2)
C
               IF(SIGN1.NE.SIGN2)THEN
C
C THEN OPPOSITE SIGNS FOUND CHECK THE REAL PARTS
C
                  RMAX1=MAX(ROOTS(1,1),ROOTS(1,2))
C
C CHECK FOR A NONZERO VALUE
C
                  IF(RMAX1.LT.SMALL)THEN
```

```
C
C THE COMPONENTS ARE EQUAL
C
C SET THE ONE REAL ROOT AS THE THIRD VALUE
C
                    NREAL=1
                    RROOTS(1)=ROOTS(1,3)
                    GOTO100
C
                  ELSE
C
C SET THE NORMS
C
                    XNORM1=ROOTS(1,1)/RMAX1
                    XNORM2=ROOTS(1,2)/RMAX1
C
                    IF(ABS(XNORM1-XNORM2).LT.OTEN)THEN
C
C SET THE ONE REAL ROOT TO THE THIRD VALUE
C
                      NREAL=1
                      RROOTS(1)=ROOTS(1,3)
                      GOTO100
C
                    ENDIF
                  ENDIF
                ENDIF
              ENDIF
C
C
C CHECK FOR EQUAL VALUES AND OPPOSITE SIGN OF 1 AND 3
C
            IF(ABS(ABS(XNORM1)-ABS(XNORM3)).LT.OTEN)THEN
C
C CHECK FOR OPPOSITE SIGNS
C
              SIGN1=SIGN(ONE,XNORM1)
              SIGN2=SIGN(ONE,XNORM3)
C
              IF(SIGN1.NE.SIGN2)THEN
C
C THEN OPPOSITE SIGNS FOUND CHECK THE REAL PARTS
C
                RMAX1=MAX(ROOTS(1,1),ROOTS(1,3))
C
C CHECK FOR A NONZERO VALUE
C
                IF(RMAX1.LT.SMALL)THEN
C
C THE COMPONENTS ARE EQUAL
C
C SET THE ONE REAL ROOT AS THE THIRD VALUE
C
                  NREAL=1
                  RROOTS(1)=ROOTS(1,2)
                  GOTO100
C
                ELSE
C
C SET THE NORMS
C
                  XNORM1=ROOTS(1,1)/RMAX1
```

```
                    XNORM2=ROOTS(1,3)/RMAX1
C
                    IF(ABS(XNORM1-XNORM2).LT.OTEN)THEN
C
C SET THE ONE REAL ROOT TO THE THIRD VALUE
C
                       NREAL=1
                       RROOTS(1)=ROOTS(1,2)
                       GOTO100
C
                    ENDIF
                 ENDIF
              ENDIF
           ENDIF
C
C CHECK FOR EQUAL VALUES AND OPPOSITE SIGN OF 2 AND 3
C
           IF(ABS(ABS(XNORM2)-ABS(XNORM3)).LT.OTEN)THEN
C
C CHECK FOR OPPOSITE SIGNS
C
              SIGN1=SIGN(ONE,XNORM2)
              SIGN2=SIGN(ONE,XNORM3)
C
              IF(SIGN1.NE.SIGN2)THEN
C
C THEN OPPOSITE SIGNS FOUND CHECK THE REAL PARTS
C
                 RMAX1=MAX(ROOTS(1,2),ROOTS(1,3))
C
C CHECK FOR A NONZERO VALUE
C
                 IF(RMAX1.LT.SMALL)THEN
C
C THE COMPONENTS ARE EQUAL
C
C SET THE ONE REAL ROOT AS THE THIRD VALUE
C
                    NREAL=1
                    RROOTS(1)=ROOTS(1,1)
                    GOTO100
C
                 ELSE
C
C SET THE NORMS
C
                    XNORM1=ROOTS(1,2)/RMAX1
                    XNORM2=ROOTS(1,3)/RMAX1
C
                    IF(ABS(XNORM1-XNORM2).LT.OTEN)THEN
C
C SET THE ONE REAL ROOT TO THE THIRD VALUE
C
                       NREAL=1
                       RROOTS(1)=ROOTS(1,1)
                       GOTO100
C
                    ENDIF
                 ENDIF
              ENDIF
           ENDIF
C
```

```
C THERE WAS SOME NON-SMALL IMAGINARY PART BUT
C THERE WERE NO CONJUGATE ROOTS: THUS 3 REAL ROOTS
C
            NREAL=3
            RROOTS(1)=ROOTS(1,1)
            RROOTS(2)=ROOTS(1,2)
            RROOTS(3)=ROOTS(1,3)
            GOTO100
C
         ENDIF
C
      ENDIF
C
100   CONTINUE
C
C NREAL ROOTS HAVE BEEN FOUND
C ===========================
C
C DETERMINE WHICH ONE MINIMIZES THE FUNCITON
C
C
C CHECK THE THREE ROOTS AND DETERMINE THEIR FUNCITON VALUE
C ---------------------------------------------------------
C
      ROOT=1.10E10
      CMIN =1.10E10
C
      DO 200 IROOT=1,NREAL
C
C CALCULATE THE VALUE OF THE FOURTH ORDER EQN
C USING ONLY THE REAL PART
C
         X=RROOTS(IROOT)
C
         CVAL=COEF(1)+COEF(2)*X+COEF(3)*X*X+
     .                COEF(4)*X*X*X+COEF(5)*X*X*X*X
C
C SAVE OFF THE MINIMUM VALUE
C
         IF(CVAL.LT.CMIN)THEN
            ROOT=X
            CMIN=CVAL
         ENDIF
C
200   CONTINUE
C
C CHECK AND SEE OF A ROOT WAS SELECTED
C
      IF(ROOT.GT.BIG)THEN
         WRITE(LTERM,1100)
1100     FORMAT(/,/,3X,'REALROOT : REAL ROOT NOT FOUND')
      ENDIF
C
C
      RETURN
      END


      SUBROUTINE REMESH
C
C.....................................................................
```

```
C
C THIS ROUTINE IS THE DRIVER FOR THE REMESHING ALGORITHM.
C
C.....
C
C
%INCLUDE 'SYSCOM.BLK'
%INCLUDE 'PARAMS.BLK'
%INCLUDE 'LUNITS.BLK'
%INCLUDE 'BUFFER.BLK'
%INCLUDE 'CELEM.BLK'
%INCLUDE 'CNODE.BLK'
%INCLUDE 'INFLUNC.BLK'
%INCLUDE 'MESHCOEF.BLK'
C
      DIMENSION A(2,4),V(2),DV(2)
      DIMENSION OMEGA(2,3),RROOTS(3)
C
      DATA SMALL/1.0E-9/
C
C CHECK TO SEE IF THERE ARE ANY ELEMENTS THAT CAN BE REMESHED
C
      IF(NNINFR.EQ.0)RETURN
C
C CHECK TO SEE IF MESH IS TO BE REMESHED
C
      CALL CHKMESH(IFLAG)
C
C CONTINUE IF REMESHING IS REQUIRED
C
      IF(IFLAG.NE.0)THEN
C
C SET LOOP COUNTER
C
         ICOUNT=1
C
C SET THE X AND XX ARRAYS (IN COMMON BLOCK CNODE) EQUAL TO EACH OTHER
C
         CALL SETEQUAL(2*MAXND,X(1,1),XBUF(1,1))
C
C JUMP IN POINT FOR ANOTHER REMESH
C
100      CONTINUE
C
C INITIALIZE THE TOLERANCE SUM
C
         TOLSUM=0.0
C
C LOOP THROUGH INTERIOR BOUNDARY NODES
C
         DO 1000 INODE=1,NNINFR
C
            MNODE=INTNOD(1,INODE)
            V(1)=XBUF(1,MNODE)
            V(2)=XBUF(2,MNODE)
C
C FIND NEIGHBORING NODES AND LOAD THEIR COORDINATES INTO ARRAY A
C
            IBLOCK=0
            IEL=INTNOD(2,INODE)
200         IBLOCK=IBLOCK+1
C
```

```
            DO 300 N=1,4
               NTEST=NODES(N,IEL)
               IF(NTEST.EQ.MNODE)THEN
                  NNODE=N+1
                  N1=N
               ENDIF
300         CONTINUE
C
            IF(NNODE.GT.4)NNODE=NNODE-4
C
            A(1,N1)=XBUF(1,NODES(NNODE,IEL))
            A(2,N1)=XBUF(2,NODES(NNODE,IEL))
C
            NSIDE=N1-1
            IF(NSIDE.LT.1)NSIDE=NSIDE+4
            IELNEW=NELCON(NSIDE,IEL)
C
            IF(IBLOCK.LT.4)THEN
               IEL=IELNEW
               GOTO200
            ENDIF
C
C LOAD UP COEFFICIENTS FOR THE REMESHING EQUATIONS
C
            CALL LOADCOF1(A(1,1))
C
C CALCULATE THE DENOMINATOR OF BETA TO DETERMINE IF IT IS ZERO
C
            DENOM=APN(INODE,1)*APN(INODE,1)
                 +APN(INODE,2)*APN(INODE,2)

C CALCULATE DELTA V
C
            IF(ICOUNT.EQ.1.OR.DENOM.EQ.0.0)THEN
C
C LOAD DUMMY ARRAY APN WITH DELTA V(0)
C
               APN(INODE,1)=-(C(2)+2.0*C(4)*V(1)+C(6)*V(2)
                    +3.0*C(7)*V(1)*V(1)+2.0*C(8)*V(1)*V(2)
                    +C(7)*V(2)*V(2)+4.0*C(9)*V(1)*V(1)*V(1)
                    +2.0*C(10)*V(1)*V(2)*V(2))
               APN(INODE,2)=-(C(3)+2.0*C(5)*V(2)+C(6)*V(1)
                    +3.0*C(8)*V(2)*V(2)+2.0*C(7)*V(1)*V(2)
                    +C(8)*V(1)*V(1)+4.0*C(9)*V(2)*V(2)*V(2)
                    +2.0*C(10)*V(1)*V(1)*V(2))
C
               PN(INODE,1)=APN(INODE,1)
               PN(INODE,2)=APN(INODE,2)
C
            ELSE
C
C CALCULATE BETA
C
C LOAD DUMMY ARRAY PN WITH GRAD F(V(N))
C
               PN(INODE,1)=-(C(2)+2.0*C(4)*V(1)+C(6)*V(2)
                    +3.0*C(7)*V(1)*V(1)+2.0*C(8)*V(1)*V(2)
                    +C(7)*V(2)*V(2)+4.0*C(9)*V(1)*V(1)*V(1)
                    +2.0*C(10)*V(1)*V(2)*V(2))
               PN(INODE,2)=-(C(3)+2.0*C(5)*V(2)+C(6)*V(1)
                    +3.0*C(8)*V(2)*V(2)+2.0*C(7)*V(1)*V(2)
                    +C(8)*V(1)*V(1)+4.0*C(9)*V(2)*V(2)*V(2)
```

```
                    +2.0*C(10)*V(1)*V(1)*V(2))
                  BETA=(PN(INODE,1)*PN(INODE,1)
                       +PN(INODE,2)*PN(INODE,2))/DENOM
C
C LOAD DUMMY ARRAY APN WITH DELTA V(N)
C
                  APN(INODE,1)=PN(INODE,1)+BETA*APN(INODE,1)
                  APN(INODE,2)=PN(INODE,2)+BETA*APN(INODE,2)
C
               ENDIF
C
C LOAD THE G COEFFICIENTS
C
               DV(1)=APN(INODE,1)
               DV(2)=APN(INODE,2)
C
               CALL LOADCOF2(V(1),DV(1))
C
C DETERMINE IF THE COEFFICIENT ON THE OMEGA-CUBED TERM IS ZERO
C
               IF(G(5).EQ.0.0)THEN
C
C DETERMINE IF THE COEFFICIENT ON THE OMEGA-SQUARED TERM IS ZERO
C IF SO, DETERMINE IF THE EQUATION IS LINEAR AND SOLVE FOR WN
C IF IT IS NOT LINEAR, SET WN EQUAL TO ZERO
C
                  IF(G(4).EQ.0.0)THEN
                     IF(G(3).NE.0.0)THEN
                        WN=-G(2)/(2.0*G(3))
                     ELSE
                        WN=0.0
                     ENDIF
                     GOTO700
C
                  ENDIF
C
C IF ONLY THE OMEGA-CUBED COEFFICIENT IS ZERO SOLVE USING
C THE QUADRATIC EQUATION
C
                  G1=3.0*G(4)
                  G2=2.0*G(3)
                  G3=G(2)
C
C DETERMINE IF THE RADICAL IS POSITIVE OR NEGATIVE
C
                  RAD=G2*G2-4.0*G1*G3
                  IF(RAD.LT.0.0)THEN
                     WN=0.0
                  ELSE
                     ROOT=SQRT(RAD)
                     W1=(-G2+ROOT)/G1
                     W2=(-G2-ROOT)/G1
C
C IF THE RADICAL IS POSITIVE, TEST THE ROOTS FOR MINIMIZATION OF PSI
C
                     PTEST1=G(1)+G(2)*W1+G(3)*W1*W1+G(4)*W1*W1*W1
                     PTEST2=G(1)+G(2)*W2+G(3)*W2*W2+G(4)*W2*W2*W2
                     IF(PTEST1.LE.PTEST2)THEN
                        WN=W1
                     ELSE
                        WN=W2
                     ENDIF
```

```
C
                ENDIF
C
            ELSE
C
C MINIMIZE THE PSI EQUATION WITH RESPECT TO OMEGA
C   PSI=F(V(N)+OMEGA*DELTA V(N))
C
                G1=3.0*G(4)/(4.0*G(5))
                G2=2.0*G(3)/(4.0*G(5))
                G3=G(2)/(4.0*G(5))
C
                CALL ZCUBIC(G1,G2,G3,OMEGA,IDUMMY)
C
                CALL REALROOT(G,OMEGA,IDUMMY,WN)
C
            ENDIF
C
C LOAD OMEGA(N)*DELTA V(N)  INTO DUMMY ARRAY DN
C
700         DN(INODE,1)=WN*APN(INODE,1)
            DN(INODE,2)=WN*APN(INODE,2)
C
C ADD GRAD F TO THE TOLERANCE SUM
C
            TOLSUM=TOLSUM+PN(INODE,1)*PN(INODE,1)
                  +PN(INODE,2)*PN(INODE,2)
C
1000      CONTINUE
C
C UPDATE THE COORDINATES
C
          DO 1200 INODE=1,NNINFR
C
             MNODE=INTNOD(1,INODE)
             XBUF(1,MNODE)=XBUF(1,MNODE)+DN(INODE,1)
             XBUF(2,MNODE)=XBUF(2,MNODE)+DN(INODE,2)
C
1200      CONTINUE
C
C DETERMINE IF ANOTHER ITERATION IS NECESSARY
C
          TOLSUM=SQRT(TOLSUM)
          IF(TOLSUM.GT.1.0E-7*NNINFR)THEN
             IF(ICOUNT.LT.25)THEN
                ICOUNT=ICOUNT+1
                IF (ICOUNT .EQ. 25) THEN
                   WRITE(LTERM,*)'TOLSUM = ',TOLSUM
                ENDIF
                GOTO 100
             ENDIF
          ENDIF
C
C AFTER ITERATION INTERPOLATE THE VALUES AT THE NEW LOCATIONS
C
          CALL INTERPOL
C
C AFTER THE NEW VALUES HAVE BEEN EVALUATED, PUT THE NEW NODAL
C COORDINATES INTO THE X COORDINATE ARRAY
C
          DO 1500 INODE=1,NNINFR
             MNODE=INTNOD(1,INODE)
```

```
            X(1,MNODE)=XBUF(1,MNODE)
            X(2,MNODE)=XBUF(2,MNODE)
1500      CONTINUE
C
      ENDIF
C
C
      RETURN
      END

      THE FOLLOWING COMMON BLOCKS ARE USED IN THE REMESHING ROUTINES

      COMMON /BUFFER/ APN(MAXND,4),PN(MAXND,4),DN(MAXND,4),
     .                XBUF(2,MAXND),INTBUF(2,MAXND),INTV(MAXND)
C
C PARAMETERS:
C     APN     - DUMMY ARRAY USED IN THE SOLVER AND ELSEWHERE
C     PN      - DUMMY ARRAY
C     DN      - DUMMY ARRAY
C     XBUF    - DUMMY ARRAY USED IN REMESHING
C     INTBUF  - DUMMY ARRAY
C     INTV    - DUMMY ARRAY


      COMMON /CCON/ NNODE,NELEM,NUMGRP
C
C PARAMETERS:
C     NNODE     - NUMBER OF NODES
C     NELEM     - NUMBER OF ELEMENTS
C


      COMMON /CELEM/ NODES(4,MAXEL),NELCON(8,MAXEL),NELGRP(4,MAXEL)
C
C PARAMETERS:
C     NODES     - ELEMENT NODE NUMBERS
C     NELCON    - ELEMENT CONNECTIVITY ARRAY
C


      COMMON /CNODE/ X(2,MAXND)
C
C PARAMETERS:
C     X       - COORDINATES ARRAY
C


      COMMON /INFLUNC/ RADINF,VELN(2,MAXND),
     .                 NNINFB,NNINFR,INFBND,INFNOD(MAXBC),
     .                 INTNOD(2,MAXND)
C
C PARAMETERS:
C     RADINF - RADIUS OF INFLUENCE FOR TIME DEPENDENT GRIDS
C     VELN   - NODAL VELOCITIES
C     NNINFB - NUMBER OF NODES ON THE INFLUENCE BOUNDARY
C     NNINFR - NUMBER OF NODES IN THE INFLUENCE REGION
C     INFBND - INFLUENCE BOUNDARY NUMBER OF THE MOVING BOUNDARY
C     INFNOD - LIST OF NODES ON THE INFLUENCE BOUNDARY
C     INTNOD - (1) LIST OF NODES IN THE INFLUENCE REGION
C              (2) ELEMENT ASSOCIATED WITH NODE
C
```

```
      COMMON /LUNITS/ LTERM
C
C
C PARAMETERS:
C
C     LTERM - TERMINAL SCREEN UNIT NUMBER  = 7
C


      COMMON /MESHCOEF/C(10),G(5),ALPHA,TOLMESH,TOLMESH1
C
C PARAMETERS:
C     C         - COEFFICIENTS FOR THE F FUNCTION AND ITS GRADIENT
C     G         - COEFFICIENTS FOR THE PSI FUNCTION AND ITS DERIVATIVE
C     ALPHA     - WEIGHT PARAMETER TO TRADE OFF SMOOTHNESS VS.
C                 ORTHOGONALITY (F=ALPHA*ORT+(1-ALPHA)*SM)
C     TOLMESH   - TOLERANCE VALUE FOR REMESHING
C     TOLMESH1  - TOLERANCE VALUE FOR REMESHING
C
C THESE COEFFICIENTS ARE USED IN THE REMESHING ROUTINES
C


      COMMON/MOVE/ IELACT(MAXEL),NODACT(MAXND)
C
C
C     IELACT(I)- FLAG INDICATING THE LOCATION OF AN ELEMENT I
C                =  0 INACTIVE
C                =  1 ACTIVE
C                =  2 ACTIVE AND IN THE INFLUENCE REGION
C                =  3 ACTIVE AND BOUNDING THE MOVING PORTION OF THE
C                     INFLUENCE REGION
C     NODACT    - FLAG INDICATING WHETHER A NODE IS ACTIVE OR NOT
C                 = 0 INACTIVE
C                 = 1 ACTIVE


      PARAMETER (IELMAX=250)
      PARAMETER (MAXEL=10000)
      PARAMETER (MAXND=10000)
      PARAMETER (MXNBC=10)
      PARAMETER (MAXBC=800)
      PARAMETER (MXNDF=4)
      PARAMETER (MAXST1=1500)
      PARAMETER (MAXST2=3000)
C
C PARAMETERS:
C     IELMAX - MAXIMUM NUMBER OF ELEMENTS TO BE 'RECOMPUTED'
C     MAXEL  - MAXIMUM NUMBER OF ELEMENTS
C     MAXND  - MAXIMUM NUMBER OF NODES
C     MXNBC  - MAXIMUM NUMBER OF BOUNDARY CONDITIONS
C     MAXBC  - MAXIMUM NUMBER OF BCS
C     MAXST1 - MAXIMUM NUMBER OF NODES/ELEMENTS FOR SOLID REGION
C     MAXST2 - TWO TIMES MAXST1
C


      COMMON /QUADR/ XQ1(2,4),XBNDRY(2,8)
C
C PARAMETERS:
C     XQ1     - GAUSS INTEGRATION POINT COORDS IN THE MASTER ELEMENT
```

```
C     XBNDRY - BOUNDARY GAUSS INTEGRATION POINTS
C


      COMMON /SOLVEC/ QTN(MAXND,4),QTN1(MAXND,4)
C
C PARAMETERS:
C     QTN    - SOLUTION VECTOR
C     QTN1   - SOLUTION VECTOR AT THE PREVIOUS STEP
C


      COMMON /TRANSFO/ XMO,YMO,CXXO,CXHO,CXXHO,CYXO,CYHO,CYXHO,
                       XCO(4),YCO(4),IEL,LINT
C
C PARAMETERS:
C     XMO    - AVERAGE ELEMENT X COORDS
C     YMO    - AVERAGE ELEMENT Y COORDS
C     CXXO   - (X(2)-X(1)+X(3)-X(4))/4
C     CYXO   - (Y(2)-Y(1)+Y(3)-Y(4))/4
C     CXHO   - (X(3)-X(2)+X(4)-X(1))/4
C     CYHO   - (Y(3)-Y(2)+Y(4)-Y(1))/4
C     CXXHO  - (X(1)-X(2)+X(3)-X(4))/4
C     CYXHO  - (Y(1)-Y(2)+Y(3)-Y(4))/4
C     XCO    - NODAL X COORDINATES FOR ELEMENT IEL
C     YCO    - NODAL Y COORDINATES FOR ELEMENT IEL
C     IEL    - ELEMENT NUMBER
C     LINT   - INTEGRATION POINT NUMBER
C
C NOTE: THESE PARAMETERS ARE USED TO DETERMINE THE JACOBIANS
C
C
```